Constant Complements, Reversibility and Universal View Updates*

Michael Johnson[1] and Robert Rosebrugh[2]

[1] Department of Computer Science
Macquarie University
mike@ics.mq.edu.au
[2] Department of Mathematics and Computer Science
Mount Allison University
rrosebrugh@mta.ca

Abstract. The algebraic specification of information systems (including databases) has been advanced by the introduction of category theoretic sketches and in particular by the authors' Sketch Data Model (SkDM). The SkDM led to a new treatment of view updating using universal properties already studied in category theory. We call the new treatment succinctly "universal updating". This paper outlines the theory of universal updating and studies the relationships between it and recent theoretical results of Hegner and Lechtenbörger which in turn studied the classical "constant complement" approach to view updates. The main results demonstrate that constant complement updates are universal, that on the other hand there are sometimes universal updates even in the absence of constant complements, and that in the SkDM constant complement updates are reversible. We show further that there may be universal updates which are reversible even for views which have no complement. In short, the universal updates provide an attractive option including reversibility, even when constant complements are not available. The paper is predominantly theoretical studying different algebraic approaches to information system software but it also has important practical implications since it shows that universal updates have important properties in common with classical updates but they may be available even when classical approaches fail.

Keywords: View update, semantic data model, category theory.

1 Introduction

To provide usability, security, access limitation, and even interoperability for database systems, the designer of a database schema may specify a subschema or "view". Any database state instantiating the database schema determines a view state instantiating the view schema by substitution. A user with access to the view state may perform an update to the view state. The question arising

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

J. Meseguer and G. Roşu (Eds.): AMAST 2008, LNCS 5140, pp. 238–252, 2008.

is how to determine an appropriate update to the state of the total database. This problem, known as the "view update problem" has been widely studied. There is a variety of "solutions", referred to as "translations", but many of these are either narrow in their application or not apparently close to actual database models. The implementation of view updates, especially within standards such as SQL, has been largely based on ad hoc and very limiting requirements.

Much of the literature on the view update problem is over 15 years old, but in recent years there have been several new contributions. In 2002, Hegner [6] introduced an order based theory of database views and their updates which generalized the constant complement approach to view updating originally developed by Bancilhon and Spyratos [1]. In 2003 Lechtenbörger [10] explored the relationship between the reversibility of updates and constant complements. More recently Bohannon, Pierce and Vaughan [4] introduced *lenses*, a structure providing a lifted state for a given state and the updated version of its view state. Lenses guarantee translations for the constant complement views and they noted that view updating in the style of Bancilhon and Spyratos allows only a "relatively small number of updates on a view to be translated". Dayal and Bernstein [5] were more permissive in the view update translations that they proposed and also considered a criterion that in modern terms would be described as a *universal property*: They discuss (p 401) the desire for view update translations to be *unique* and *minimal*. In a similar vein, Hegner finds that "within the constant complement order-based framework, the reflection of an order based update of a view to the base schema is unique, without qualification." The present authors have investigated an approach to database schemas, states and views based on categorical algebra [7]. This data model prescribes a solution to the view update problem using precisely universal properties (unique minimal translations).

The problem addressed by this paper is to understand better the relationship between universal updates and constant complement updates in the context of our data model. Hegner's order-based context models database states more accurately than considering them to be abstract sets, and his results are suggestive of what we will find. Lechtenbörger showed that, in a suitable sense, constant complement updates were always reversible, and conversely if all updates to a view are reversible then it is possible to find a constant complement for it. The main contributions of the present article are

1. To develop the framework in which universal updates properly reflect ambient database structure
2. To show that in that framework constant complement updates are, in harmony with Hegner's results, necessarily universal
3. To show that in that framework constant complement updates are, in harmony with Lechtenbörger's results, reversible
4. To provide examples to demonstrate that universal updates are more general than both constant complement updates and reversible updates

We note particularly that view updates can have very attractive properties including universality and even reversibility without necessarily having any constant complement (without contradicting Lechtenbörger's results, see below).

As mentioned above, the work presented here uses the *sketch data model (SkDM)* [8], [9] which is based on categorical algebra. This data model is related to both the popular and widely used entity-relationship model and to the functional data model of Shipman [12]. Entities and attributes are modelled using a simple graphical language. Relationships among them are constrained using concepts from category theory that express selection, projection, join and union constraints. The syntactic formalism derived from these ideas is expressed by the concept of "sketch". The sketches we use are called Entity-Attribute (EA)-sketches and are described in detail below.

Straightforward procedures and a variety of software applications translate entity-relationship diagrams into relational database schemas, whose semantics are database states. For the sketch data model, a similar implementation is in progress (see for example [11], an EA-sketch compiler which supports graphical manipulation of EA-sketches and automatic conversion into Oracle and MySQL database schemas).

The structure of the paper is as follows. In Section 2 we present a small motivating example of a sketch data model followed by the formal definitions of EA sketches, views, and propagatable (universal) updates. Section 3 is devoted to developing the main results relating propagatable updates to constant complements and reversibility respectively. Finally, Section 4 relates those main results to the work of others.

2 An Example and the Sketch Data Model

The results presented in Section 3 need the technical details of the sketch data model which are given below. First we work through a motivating example. Other examples can be found in [9] and in case-studies and consultancies, cited there.

We assume some familiarity with Entity-Relationship (ER) notation and the basic language of categories and functors applied to Computer Science as found, for example, in [2].

Example 1. When aircraft land at an airport in restricted visibility conditions they use an "instrument approach". We describe part of a database schema for instrument approaches. The main entities involved are airports, runways, waypoints (fixes) and the instrument approaches themselves. For example, there is a VOR (VHF Omni-Range) approach to Runway 18 at Willard Airport near Champaign. An approach also involves a specified waypoint of one of several types which defines its final approach fix (faf). The main types of waypoints are VOR, NDB (another kind of navigation aid) and GPS waypoints (GPSWP). Also required is a waypoint to fly to in case of a missed approach (overshoot).

An EA sketch has four components which we will now prescribe for our example. The first component (as for ER diagrams) is a directed graph G. Figure 1 shows (the main elements of) the directed graph for an EA sketch.

Entities are nodes of G, but there are no "relationship" nodes. In an ER diagram Approach might be a relationship from Runway to Waypoint. Here that

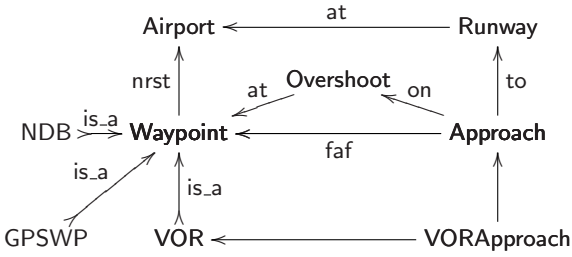


Fig. 1. Graph for part of an instrument approach database schema

is expressed by the directed edges to and faf. In a database state an entity node is modelled by an entity set just as for an ER diagram. Instead of being modelled as a relationship set in the ER fashion, Approach is also modelled as an entity set. However, the directed edges from Approach are modelled by functions. The is_a relationships are denoted here by edges indicated $\rightarrowtail$. As for an ER diagram, they are modelled by injective mappings. The other directed edges in G are modelled by functions (and can be thought of as methods or, from the database perspective, as integrity constraints). Given an instance of their source type they return an instance of their target type.

The other three components of an EA sketch do not appear in ER diagrams and they express database constraints. The second component is a set of *commutative diagrams.* Here a commutative diagram is a pair of paths in G with common source and target. They specify equality functional constraints. In our example, the right-then-top and bottom-then-left paths around the upper rectangle is a commuting diagram. It represents a real-world constraint: Each (instrument) Approach to a particular Runway at a particular Airport uses as faf a Waypoint located at that same airport. In contrast, the two paths from Approach to Waypoint (the triangle) is *not* a commutative diagram—the rules for a particular approach require that on overshooting, an aircraft holds at a particular Waypoint which will not usually be the final approach fix Waypoint. In this example, the bottom rectangle is also a commutative diagram as noted below.

The last two components of an EA sketch also express database constraints. They will require a node of G to have an entity set in a database state that depends on those of other nodes for the state. The third component of an EA sketch is a set of *finite cones* in G. A cone in G has a vertex, a base diagram, and projection edges from the vertex to base nodes. An example from Figure 1 follows. This cone has *vertex* VORApproach (the vertex is a node of G). The *base diagram* of the cone is the pair of edges VOR $\longrightarrow$ Waypoint $\longleftarrow$ Approach. The base is a diagram in the graph, given formally by a graph morphism to G. The *projections* from the vertex to the base nodes are the edges in G from VORApproach to Approach, VOR and Waypoint—the last edge is not shown in Figure 1, but it is the common value of the right-then-top and bottom-then-left paths (since the bottom rectangle is commutative!) The constraint this cone expresses is that in models VORApproach is the pullback of the base cospan. In

fact this is precisely how join database constraints are specified. Selection, is_a and projection constraints can also be expressed with finite cones. We mention a further point about the cones. An EA sketch is required to have a special cone whose vertex is called 1 (and usually not depicted) and whose base diagram is empty so that in models its value is 1.

The fourth component of an EA sketch is a set of *finite discrete cocones*. A discrete cocone has a vertex, a base diagram and links *to* the vertex *from* base nodes. Being *discrete* means no edges are permitted in the base diagram. In Figure 1 there is a discrete cocone with vertex Waypoint. Its base nodes are NDB, VOR, and GPSWP. The links to the vertex from the base nodes are the is_a edges. For discrete cocones the links are called *injections*; they are necessarily injective functions in database states. This cocone expresses the constraint that the elements of (the entity set) Waypoint are exactly the disjoint union of (the elements of the entity sets) NDB, VOR, and GPSWP. The formal requirement is that the vertex be the *sum*, or coproduct, of the base nodes.

As is usual practice, we did not draw the attributes—the "A" in EA—in Figure 1, but they are definitely a part of the EA sketch. Attributes are (often large) fixed value sets. Examples in this case are the radio frequency of a navigation aid, the surface type of a runway, the length of a runway, the four character identifying code of an airport, etc. An attribute is the vertex of a cocone whose finite discrete base has all of its nodes specified by the special node 1 and whose injection edges are called *elements*. In every state, an attribute's value is exactly the sum of its elements. Formally, the cocones that define attributes are part of the underlying graph. (In practice, attributes are usually listed separately in a data dictionary.)

We now proceed with the formalism required for EA sketches and their model categories. The first three definitions are general [2] and are included to establish notation before we specialize to our EA sketches and their models.

Definition 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 directed 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* $C(\mathbb{E})$*.*

Definition 2. *Let* $\mathbb{E} = (G, \mathbf{D}, \mathcal{L}, \mathcal{C})$ *and* $\mathbb{E}' = (G', \mathbf{D}', \mathcal{L}', \mathcal{C}')$ *be sketches. A* sketch morphism $h : \mathbb{E} \to \mathbb{E}'$ *is a graph morphism* $G \to G'$ *which carries, commutative diagrams in* $\mathbf{D}$*, cones in* $\mathcal{L}$ *and cocones in* $\mathcal{C}$ *to respectively commutative diagrams in* $\mathbf{D}'$*, cones in* $\mathcal{L}'$ *and cocones in* $\mathcal{C}'$*.*

Definition 3. *Denote the category of finite sets by* $\mathbf{set}_f$*. A* model M *of a sketch* $\mathbb{E}$ *is a functor* $M : C(\mathbb{E}) \to \mathbf{set}_f$ *such that the cones in* $\mathcal{L}$ *and cocones in* $\mathcal{C}$ *are sent to limit cones and colimit cocones in* $\mathbf{set}_f$*. If* M *and* M' *are models of* $\mathbb{E}$ *a* morphism $\phi : M \to M'$ *is a natural transformation from* M *to* M'*. The category* $\mathrm{Mod}(\mathbb{E})$ *has objects the models of* $\mathbb{E}$ *and arrows the morphisms of models.*

EA sketches as described in Example 1 have some limitations on their cones and cocones and they are adequate for describing the needed database constraints, but restrictive enough to permit definition of the query language.

Definition 4. *An* EA sketch $\mathbb{E} = (G, \mathbf{D}, \mathcal{L}, \mathcal{C})$ *is a sketch with only finite cones and finite discrete cocones and with a specified cone with empty base whose vertex is called* 1*. Edges with domain* 1 *are called* elements*. Nodes which are vertices of cocones all of whose injections are elements are called* attributes*. Nodes which are neither attributes, nor* 1*, are called* entities*.*

The next definitions are fundamental: the semantics i.e. database states, for an EA sketch are specified by a set for each node of the graph and a function for each edge subject to: equality of composition constraints (from the commutative diagrams); select, project and join constraints (from the cones); and (disjoint) union constraints (from the discrete cocones).

Definition 5. *A* database state *D for an EA sketch $\mathbb{E}$ is a model of $\mathbb{E}$. The* category of database states of $\mathbb{E}$ *is* $\mathrm{Mod}(\mathbb{E})$*. An* insert update *(respectively* delete update*) for a database state D is a monomorphism $D \rightarrowtail D'$ (respectively $D' \rightarrowtail D$) in* $\mathrm{Mod}(\mathbb{E})$*.*

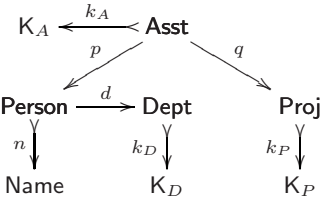
A morphism of database states is a monomorphism when each component morphism is monic. Thus our definition of a delete (resp. insert) update means that some elements are deleted (resp. inserted) in in the set specified for each node. The following definition encodes the requirement of the relational model for *entity integrity*, since it means that there is a chosen primary key for each entity.

Definition 6. *The EA sketch $\mathbb{E}$ is* keyed *if for each entity E there is a specified attribute A_E called its* key attribute *and a chosen monic specification $E \rightarrowtail A_E$.*

For a keyed EA sketch, it turns out that all morphisms between database states are monomorphisms ([8], Proposition 4.7) since all of the natural transformation component mappings are injective.

Example 2. We give a simple EA sketch $\mathbb{E}_1$ that we will consider in the sequel. It is a variant of an example in [6]. The database records people, their names and departments and their assignments to projects.

The graph is just that depicted below. There are no commutative diagrams



The attributes are K_A, Name, K_D and K_P (we do not show their cocone specifications in the diagram). Entities are Asst, Person, Dept and Proj. The edges k_A, n, k_D and k_P are keys, so there are also several cones not shown. ◇

Before giving the definition of view, we briefly discuss the *query language* of an EA sketch. One of the advantages of the sketch data model is that an EA sketch comes equipped with a query language. For any EA sketch $\mathbb{E}$ there is a category called the *theory* of the sketch denoted $Q(\mathbb{E})$ (for details consult [3, Section 8.2]). This $Q(\mathbb{E})$ is constructed starting from $C(\mathbb{E})$ and then formally adding to it all finite limits and finite sums, subject to the (co)cones in $\mathcal{L}$ and $\mathcal{C}$. For example, $Q(\mathbb{E}_1)$ has objects like $\mathsf{Pers} \times \mathsf{Proj}$ and $\mathsf{Dept} + \mathsf{Asst}$. From its construction, $Q(\mathbb{E})$ includes $\mathbb{E}$ and actually has *all* finite limits and finite sums. An essential point is that $\mathrm{Mod}(Q(\mathbb{E}))$ is equivalent as a category to $\mathrm{Mod}(\mathbb{E})$. This is because a $Q(\mathbb{E})$ model restricts to an $\mathbb{E}$ model and conversely an $\mathbb{E}$ model determines values on queries and so a $Q(\mathbb{E})$ model.

A *view* allows a user of an information system to manipulate data which are part of, or are derived from, an underlying database. As we are about to define it, a view of an EA sketch $\mathbb{E}$ has a new EA sketch $\mathbb{V}$ with the entities of $\mathbb{V}$ interpreted via a sketch morphism V as entities from the original EA sketch $\mathbb{E}$ or even query results (entities of $Q(\mathbb{E})$). Formally,

Definition 7. *A* view *of an EA sketch* $\mathbb{E}$ *is an EA sketch* $\mathbb{V}$ *together with a sketch morphism* $V : \mathbb{V} \to Q(\mathbb{E})$.

Example 3. A view $V_1 : \mathbb{V}_1 \longrightarrow Q(\mathbb{E}_1)$ of $\mathbb{E}_1$ is specified by the inclusion in $Q(\mathbb{E}_1)$ of the sketch whose graph is just the three edges k_A, np and $k_P q$ (the latter two being composites of edges in $\mathbb{E}_1$). Note that the composite edges np and $k_P q$ are not edges in $\mathbb{E}_1$ but they are present in $C(\mathbb{E}_1)$ and so in $Q(\mathbb{E}_1)$. $\diamond$

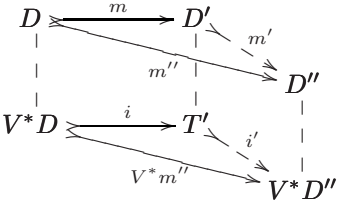
The equivalence of $\mathrm{Mod}(\mathbb{E})$ with $\mathrm{Mod}(Q(\mathbb{E}))$ means a database state $D : \mathbb{E} \to \mathbf{set}_f$ can also be considered as a model of $Q(\mathbb{E})$, also denoted D. Composing the model D with a view V gives a database state $DV : \mathbb{V} \to Q(\mathbb{E}) \to \mathbf{set}_f$ for $\mathbb{V}$, the *V-view of D*. This operation of *composing with V* is written V^* so $V^*D = DV$ and so we obtain a functor $V^* : \mathrm{Mod}(\mathbb{E}) \to \mathrm{Mod}(\mathbb{V})$ which sends a database state for $\mathbb{E}$ to one for $\mathbb{V}$.

We sometimes refer to a database state of the form V^*D as a "view". Context determines whether "view" refers to a database state, or to the sketch morphism V. To avoid ambiguity we also refer to V^*D as a *view state* and to $\mathbb{V}$ as the *view sketch*. Our framework implies an inessential difference from other work on views. When the database states are simply a set [1], [10] instead of a category like $\mathrm{Mod}(\mathbb{E})$ the analogue of V^* is a mapping called the *view definition mapping*. If the states are a partially ordered set [6] it is a monotone mapping. These mappings are usually required to be surjective so that every view state arises from a state of the underlying database. While V^* may be surjective on objects (view states), we do not require this, so we formally allow view states not derived from underlying database states. In examples V is usually one-one on objects making V^* surjective on objects.

Since a view state is itself a database state for its view sketch, we may (subject to the constraints of the view sketch) insert items in or delete items from a view state. An insertion in, or deletion from, the view state V^*D is translatable to the underlying database state D if there is an insertion in or deletion from

the underlying database which, on application of V^*, becomes the given view insert/delete. We will say the insertion or deletion is *propagatable* if there is a unique "minimal" insert/delete in the following sense.

Definition 8. *Let $V : \mathbb{V} \to Q(\mathbb{E})$ be a view of $\mathbb{E}$. Suppose D is a database state for $\mathbb{E}$, T' a database state for $\mathbb{V}$, and $i : V^*D \rightarrowtail T'$ is an insert update of V^*D. The insertion i is* propagatable *if there exists an insert update $m : D \rightarrowtail D'$ in* Mod($\mathbb{E}$) *with the following property: $i = V^*m$ and for any database state D'' and insert update $m'' : D \rightarrowtail D''$ such that $V^*m'' = i'i$ for some $i' : T' \rightarrowtail V^*D''$, there is a unique insert $m' : D' \rightarrowtail D''$ such that $V^*m' = i'$ as in*



where the dashed vertical lines indicate, for example, that V^*D is the image of D under V^*. If every insert update on V^*D is propagatable, we say that the view state V^*D is insert updatable.

Definitions of *propagatable* for a deletion $d : T' \rightarrowtail V^*D$ and *delete updatable* for a view state are obtained by reversing some arrows. Note that the m whose existence is required is *essentially unique*. By this we mean that for any other $n : D \rightarrowtail E'$ that satisfies the requirements on m, there is an invertible morphism of database states $j : D' \to E'$ satisfying $jm = n$.

Example 4. Let D be a model of $\mathbb{E}_1$. We consider some updates for the view V_1 of Example 3.

First, any delete from the value $V_1^*D(\mathsf{Asst})$ is propagatable: the deleted assignment is simply deleted from $D(\mathsf{Asst})$. There are no other consequences. Since the other nodes of the graph of $\mathbb{V}_1$ are attributes and hence their values are the same in every database state, no other deletes are possible.

Next consider inserting an item a into the view state V_1^*D at $V_1^*D(\mathsf{Asst})$. That is, we wish to add an assignment in the view. Once again, there are no other possibilities for insertion in V_1^*D. This requires defining $V_1^*D \rightarrowtail T'$ and so defining K_A, Name and K_P values $T'(k_A)(a)$ and so on. If the proposed Name and K_P values, $T'(np)(a)$ and $T'(k_Pq)(a)$ are already in the images of $D(n)$ and $D(k_P)$ because the person and project exist already, and provided there is a free assignment key in $D(K_A)$ for the value of $T'(k_A)(a)$, then the insert is propagatable to a $D \rightarrowtail D'$ since values of $D'(p)(a)$ (and hence also $D'(d)(a)$) and $D'(q)(a)$ are determined. Even if the proposed K_P value is not an image of $D(k_P)$ this remains true since it is then possible to insert an item b into $D(\mathsf{Proj})$ and set $D'(q)(a) = b$ and $D'(k_P)(b) = T'(k_Pq)(a)$. However, if the proposed Name value is not an image of $D(n)$, there is no canonical choice of Dept value $D'(dp)(a)$ and consequently the update is not propagatable. ◇

When all insert (respectively, delete) updates of a view are propagatable then V^* is called a *right (resp. left) fibration*. For criteria guaranteeing this property see [9]. For historical reasons, the arrow m in Definition 8 is called *op-cartesian*, while the analogous arrow for a delete is called *cartesian* and we will use these names below.

3 The Main Results

In this section we will study constant complements and reversibility.

A notion of *view complement* appeared in the influential article of Bancilhon and Spyratos [1], a study of the view update problem. They consider database states to be a *set* S, view states to be a set V and give a surjective view definition mapping $f : S \twoheadrightarrow V$ from the database states onto the view states. A *view update* is taken to be an endo-function u on the view states. A set U of view updates is specified and assumed to be *complete*, i.e. closed under composition and including the identity function. A *translation* T_u of a view update u is a database update (endo-function on S) such that the view of a translated database state is the update of the view of the state, i.e. $f(T_u(s)) = u(f(s))$ (and T_u acts as the identity on any s whenever u acts as the identity on $f(s)$). A translation T_u is a solution to the view update problem for the update u and a *translator* T for U is a set of translations $\{T_u \mid u \in U\}$. The diagram following is suggestive:

$$\begin{array}{ccc} s & \longmapsto & T_u(s) \\ \vdots & & \vdots \\ f(s) & \longmapsto & u(f(s)) \end{array}$$

Bancilhon and Spyratos show that a translator T for a complete update set implies the existence of a "constant complement view" C. This is a second set C of view states with a second view definition mapping, say $g : S \longrightarrow C$, such that the mapping $\langle f, g\rangle : S \longrightarrow V \times C$ is a bijection (C is a *complement* of V) and such that $g(T_u(s)) = g(s)$ holds for $T_u \in T, s \in S$ (any T_u in T is *constant* on C). They also showed a converse. This is the basis of the "constant complement" update strategy.

Lechtenbörger [10] has recently shown that constant complement translators exist when all of the view updates are reversible by other view updates.

In the description above the database states are taken to be an unstructured set and (subject to completeness) the updates are simply an abstract set of endo-functions. Hegner [6] suggests that the database states and the view states ought to be partially ordered sets. Then delete and insert updates should relate states that are comparable, i.e for a state s and update u, either $s \leq u(s)$ or $u(s) \leq s$. Our definition above suggests that updates should be arrows in a category of database states. We point out an important difference. In [1] and [6] an update u acts on every (view) state—it is a process mapping states to updated states—whereas for us, an update compares a single state to an updated state.

We start with the definition that begins to express these ideas in our context.

Definition 9. *Let $\mathbb{E}$, $\mathbb{V}$ and $\mathbb{C}$ be EA sketches and $V : \mathbb{V} \longrightarrow Q(\mathbb{E})$ and $C : \mathbb{C} \longrightarrow Q(\mathbb{E})$ be views. We say C is a* complement *of V if the functor*

$$\langle V^*, C^* \rangle : \mathrm{Mod}(\mathbb{E}) \longrightarrow \mathrm{Mod}(\mathbb{V}) \times \mathrm{Mod}(\mathbb{C})$$

is full, faithful and injective on objects.

Definition 10. *Let $V : \mathbb{V} \longrightarrow Q(\mathbb{E})$ and $C : \mathbb{C} \longrightarrow Q(\mathbb{E})$ be views with C a complement of V and $\alpha : R \rightarrowtail V^*D$ be an arrow in* Mod($\mathbb{V}$). *We say that α has a* C-constant update *if there is $\hat{\alpha}$ in* Mod($\mathbb{E}$) *with $\alpha = V^*\hat{\alpha}$ and $C^*(\hat{\alpha})$ an isomorphism. Dually, α has a* C-opconstant update *if $\beta : V^*D \rightarrowtail S$ is $V^*\hat{\beta}$ and $C^*(\hat{\beta})$ is an isomorphism.*

The definition does not require that α be propagatable, but only that it be the image under V^* of some $\hat{\alpha}$. More generally,

Definition 11. *Let $V : \mathbb{V} \longrightarrow Q(\mathbb{E})$ and $C : \mathbb{C} \longrightarrow Q(\mathbb{E})$ be views with C a complement of V. We say that V has* C-constant updates *if every $\alpha : R \rightarrowtail V^*D$ has a C-constant update. V has* C-opconstant updates *is defined dually.*

Example 5. Consider an EA sketch $\mathbb{E}$ that is specified completely (except for 1) by a sum diagram

$$\mathsf{Support} \xrightarrow{i} \mathsf{Dept} \xleftarrow{j} \mathsf{Production}$$

That is, in any state the Dept entity set will be the disjoint union of the support departments and the production departments. Suppose that $\mathbb{V}$ and $\mathbb{C}$ are the EA sketches whose graphs have (in addition to 1) exactly one node each: Support and Production respectively and that $V : \mathbb{V} \longrightarrow Q(\mathbb{E})$ and $C : \mathbb{C} \longrightarrow Q(\mathbb{E})$ are the sketch morphisms providing the obvious inclusions of the two views. A state M of $\mathbb{E}$ is determined by a sum diagram:

$$M\mathsf{Support} \xrightarrow{Mi} M\mathsf{Dept} \xleftarrow{Mj} M\mathsf{Production}$$

in $\mathbf{set}_f$ and a state of either $\mathbb{V}$ or $\mathbb{C}$ by any set. Thus

$$\langle V^*, C^* \rangle : \mathrm{Mod}(\mathbb{E}) \longrightarrow \mathrm{Mod}(\mathbb{V}) \times \mathrm{Mod}(\mathbb{C})$$

sends the sum diagram above to the pair $\langle M\mathsf{Support}, M\mathsf{Production} \rangle$ of sets. It is immediate that C is a complement of V. Indeed, here $\langle V^*, C^* \rangle$ is an equivalence.

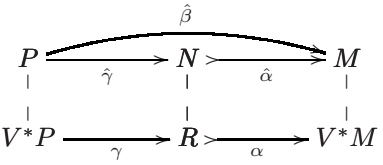
Notice that any deletion $\alpha : R \rightarrowtail V^*M$ in Mod($\mathbb{V}$) is propagatable: simply delete appropriate elements from $M\mathsf{Support}$ and the corresponding elements from $M\mathsf{Dept}$. If we denote the cartesian arrow by $\hat{\alpha} : \alpha M \longrightarrow M$, so $\alpha = V^*\hat{\alpha}$ we see that $C^*(\hat{\alpha})$ is the identity on $M\mathsf{Production}$. Thus, α has a C-constant update, and indeed V has C-constant updates. Note further that $\hat{\alpha}$ is also opcartesian for the arrow $\alpha : V^*\alpha M \rightarrowtail M\mathsf{Support}$. That is, α is 'reversible' in the sense of Definition 12.

Similarly, β has C-opconstant updates for any insertion $\beta : V^*M \rightarrowtail S$ in Mod($\mathbb{V}$), and β is reversible. ◇

The next two theorems show that the constant complement updates are among the propagatable (universal) updates.

Theorem 1. *Let $V : \mathbb{V} \longrightarrow Q(\mathbb{E})$ be a view and let $C : \mathbb{C} \longrightarrow Q(\mathbb{E})$ be a complement and $\alpha : R \rightarrowtail V^*M$ be a deletion in* Mod($\mathbb{V}$)*. If α has a C-constant update, then α is propagatable.*

Proof. Suppose that $\hat{\alpha} : N \to M$ satisfies $\alpha = V^*\hat{\alpha}$, $C^*\hat{\alpha}$ is invertible, $\hat{\beta} : P \longrightarrow M$ and $\gamma : V^*P \longrightarrow R$ satisfy $V^*\hat{\beta} = \alpha\gamma$. We need to construct a unique $\hat{\gamma} : P \to N$ satisfying $V^*\hat{\gamma} = \gamma$ as in:



Since C is α-constant, we have $\langle \gamma, (C^*\hat{\alpha})^{-1}C^*\hat{\beta}\rangle : \langle V^*P, C^*P\rangle \longrightarrow \langle V^*N, C^*N\rangle$. Now $\langle V^*, C^*\rangle$ is full by assumption, so there is an arrow $\hat{\gamma} : P \longrightarrow N$ with $V^*\hat{\gamma} = \gamma$ and $C^*\hat{\gamma} = (C^*\hat{\alpha})^{-1}C^*\hat{\beta}$.

To see that $\hat{\alpha}\hat{\gamma} = \hat{\beta}$, recall that $\langle V^*, C^*\rangle$ is faithful and note that:

$$\begin{aligned}\langle V^*, C^*\rangle\hat{\alpha}\hat{\gamma} &= \langle V^*\hat{\alpha}\hat{\gamma}, C^*\hat{\alpha}\hat{\gamma}\rangle \\ &= \langle \alpha\gamma, C^*\hat{\alpha}(C^*\alpha)^{-1}C^*\hat{\beta}\rangle \\ &= \langle V^*\hat{\beta}, C^*\hat{\beta}\rangle \\ &= \langle V^*, C^*\rangle\hat{\beta}\end{aligned}$$

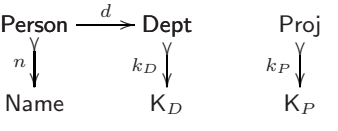
To see that $\hat{\gamma}$ is unique, just note that $\hat{\alpha}$ is monic. □

Theorem 2. *Let $V : \mathbb{V} \longrightarrow Q(\mathbb{E})$, let $C : \mathbb{C} \longrightarrow Q(\mathbb{E})$ be a complement and $\beta : V^*M \rightarrowtail S$ be an insertion in* Mod($\mathbb{V}$)*. If β has a C-opconstant update, then β is propagatable.*

Proof. Suppose that $\hat{\beta} : M \to N$ satisfies $\beta = V^*\hat{\beta}$, $C^*\hat{\beta}$ is invertible, $\hat{\alpha} : P \longrightarrow M$ and $\gamma : V^*M \longrightarrow S$ satisfy $V^*\hat{\alpha} = \gamma\beta$. The construction of $\hat{\gamma} : N \to P$ satisfying $V^*\hat{\gamma} = \gamma$ is formally dual to that above, indeed $C^*\hat{\gamma} = C^*\hat{\alpha}(C^*\hat{\beta})^{-1}$. To see that $\hat{\gamma}$ is unique, suppose that $V^*\overline{\gamma} = \gamma$ and $\overline{\gamma}\hat{\beta} = \hat{\alpha}$. Then $C^*(\overline{\gamma}\hat{\beta}) = C^*\hat{\alpha}$ so $C^*\overline{\gamma} = C^*\overline{\gamma}C^*\hat{\beta}(C^*\hat{\beta})^{-1} = C^*\hat{\alpha}(C^*\hat{\beta})^{-1} = C^*\hat{\gamma}$. Now by faithfulness of $\langle V^*, C^*\rangle$ we conclude that $\overline{\gamma} = \hat{\gamma}$. □

However, a view update may be propagatable for a view with a complement, but there may not be a constant complement.

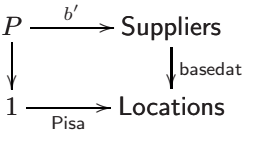
Example 6. A complement C_1 for the view V_1 of the (keyed) assignments database from Example 3 is provided by the inclusion of the sketch $\mathbb{C}_1$ whose graph is:



We saw in Example 4 that the insertion of an assignment with a new project value in a state of $\mathbb{V}_1$ can be propagatable. However, such an insertion α cannot have a C_1-constant update—its value at the entity Proj must change. Furthermore Proj must play a part in any complement, so this example shows not just that C_1 is not constant, but that no complement can be constant. ◇

Even more, a view may have all of its updates propagatable, but have no constant complement updates at all.

Example 7. Suppose that an EA sketch $\mathbb{E}$ is specified completely by commutative square with P the vertex of a cone to the right side and bottom edges (a pullback diagram):



Suppose that $\mathbb{V}$ is the EA sketch with one node P, and that V is the obvious inclusion. No choice of view complement $C : \mathbb{C} \longrightarrow Q(\mathbb{E})$ whose image is contained in $\mathbb{E}$, but which does not contain P, can provide a complement for V. To see this notice first that P specifies an inverse image (it is really a simple selection of Suppliers where based at equals Pisa). Any complement of V must contain at least the nodes Suppliers and Locations and the edge based at. If any of these is not present in a view C then $\langle V^*, C^* \rangle$ fails to be injective on objects. On the other hand, if $\mathbb{C}$ contains all of them then $\langle V^*, C^* \rangle$ is injective on objects and faithful, but it fails to be full. Indeed, for states M and M', the only arrows $\langle h, k \rangle : \langle V^*M, C^*M \rangle \longrightarrow \langle V^*M', C^*M' \rangle$ in the image of $\langle V^*, C^* \rangle$ are those where h is the induced map between inverse image values at P. ◇

This example is important since the view V has been shown [9] to be updatable universally. Indeed all inserts and deletes are propagatable for V. Even more, they are all reversible. We will show below in Theorem 3 that if a V has C-constant updates then it also has (even reversible) universal updatability, but this example shows that implication has no converse. If we modify the previous example by requiring that keys exist for P, Suppliers and Location, then the view which consists of P and its key does have a complement.

A desirable property of view updates is to be "reversible". For a view deletion this means that it is propagatable, and that the propagated deletion is also universal for the view insert which undoes the deletion. Formally,

Definition 12. *Let $V : \mathbb{V} \longrightarrow Q(\mathbb{E})$ be a view and $\alpha : R \nrightarrow V^*M$ a propagatable deletion in* Mod($\mathbb{V}$). *We say that α is* reversible *if its cartesian arrow $\hat{\alpha} : \alpha M \longrightarrow M$ is also opcartesian. Similarly, a propagatable insertion is reversible if its opcartesian arrow is also cartesian.*

Note the requirement that a deletion must be propagatable to be considered reversible. This is because talking about reversibility depends upon having chosen

an update strategy—we need to know how to propagate view updates in order to determine whether there is a view update whose propagation will undo a given update. Since we have already seen that the universal update strategy is more general than the constant complement update strategy, we base this definition on the former to provide the greatest generality. First we give an example showing that universal updatability does not guarantee reversibility.

Example 8. Consider the sketch $\mathbb{E}$ consisting of a single edge $\alpha : A \longrightarrow B$. For the two views V_A and V_B which respectively include the single node A and the single node B, it is the case that all deletes and inserts are propagatable. This is easy to see directly since V_A^* and V_B^* simply select the domain and codomain of a function (model), and it also follows from the well known fact that V_A^* and V_B^* are both left and right fibrations.

On the other hand, neither deletes nor inserts are reversible for V_A^*. Indeed, for a model, that is a function, say $f : X \longrightarrow Y$, let $V_A^* f$ be the insertion $X \rightarrowtail X + X'$. The propagated insertion is given by the model morphism from f to $f + X'$ while the propagated deletion for $X \rightarrowtail X + X' = V_A^*(f + X')$ is the (different) model morphism from jf to $f + X'$ where $j : Y \longrightarrow Y + X'$ is the injection. A similar argument shows that deletes are not reversible for V_A^*. For the case of V_B^* deletes may or may not be reversible, but it turns out that inserts are. $\diamond$

Next we show that deletions with updates that are constant in a complement are also reversible.

Theorem 3. *Let $V : \mathbb{V} \longrightarrow Q(\mathbb{E})$ and $C : \mathbb{C} \longrightarrow Q(\mathbb{E})$ be complementary views. If $\alpha : R \rightarrowtail V^*M$ is a deletion in* Mod($\mathbb{V}$) *with a C-constant update, then it is reversible. Furthermore, if V has C-constant updates , then any deletion $\alpha : R \rightarrowtail V^*M$ is reversible.*

The proof uses techniques very similar to those in Theorem 1. Lechtenbörger [10] showed that, in the context of [1] a constant complement translator implies reversibility of view updates. In that context reversibility for a set of updates means that there is an update that will undo any view update. Our definition of reversibility concerns a single propagatable update, but is similar to Lechtenbörger's.

4 Related Work and Conclusions

The article of Bancilhon and Spyratos [1] remains influential. They treat a database as an arbitrary set S, meant to specify its states, and consider a view to be an arbitrary surjective mapping $f : S \twoheadrightarrow V$. They obtain a "translator under constant complement" i.e. a function $g : S \longrightarrow C$ so that $\langle f, g\rangle$ is bijective, corresponding to our *complement*, and a translation for which $gT_u = g$, corresponding to our *constant*. The approach of Bancilhon and Spyratos has been elucidated by Bohannon et al [4]. They show that the set of translators under constant complement (for a view) corresponds to the set of "very well behaved lenses". A lens

is a (partial) view $f : C \longrightarrow A$ and a (partial) function $p : A \times C \longrightarrow C$ satisfying suitable axioms. On a database state and an *updated* view state the function p determines a database state—the updated database state for the updated view state. While it is easy to interpret a surjective mapping as a view substitution like the V^* above, the complements of [1] and [10] are projections to quotient sets. As Lechtenbörger [10] writes: "it is unexplored how such a view could be represented in SQL".

Closer to the spirit of the present work, Hegner [6] considers an ordered set D of database states. He defines a view to be a surjective monotone mapping $\gamma : D \twoheadrightarrow V$ to an ordered set V of view states such that γ reflects the order. For Hegner a "closed update family" is an order-compatible equivalence relation on the view poset. This means that related view states may be updated to one another, symmetry expressing reversibility of view updates and transitivity that they may be composed. Hegner defines an "update strategy" to be a function $V \times D \longrightarrow D$ satisfying certain axioms. It is very much like a lens, and an equivalence relation determined by the update strategy is the lifted closed update family.

We have seen that the existence of a constant complement view is sufficient to guarantee well-behaved view updatability (propagatability) and reversibility in the context of the sketch data model, but that it is by no means necessary. Either of these desirable properties may hold without complements being possible.

The work presented here has been driven by the need, motivated by industrial applications, to have definitions of view, complement, propagation, etc, which better integrate with the actual representations of databases. We have tried to move away from the idea of database states as an unstructured set, and to accurately capture the (mathematical) structures that database states bear. In addition, the theoretical operations which we develop on database states need to respect those structures.

The difference between the results presented here and those of Lechtenbörger is worth some reflection. At first sight they may appear contradictory: Lechtenbörger showed that reversibility implied the existence of a constant complement, a very appealing co-occurrence, while we showed that updates can be reversible even when no constant complement can exist. The difference of course lies in the definition of constant complement—when one requires views, and hence complements, to arise as databases with data derived from the original database (Definition 7), the extra structure limits the available views, eliminating some of the quotient views that provide some of Lechtenbörger's complements. We must emphasize that we see these limitations as positive, limiting us as they do to properly respect the actual structure of database states. Another noteworthy difference is that most earlier work studied complete sets of updates and single updates that could be applied to every possible database state (they were endofunctions on the set of database states). In contrast we have been studying the propagatability of individual insert or delete updates acting on a given state. This permits a more general treatment since we can study (universal) updates of particular states. We are currently exploring how individual universal updates

can be collected into structures corresponding to complete sets of updates so that we can further compare the new results with earlier work.

References

1. Bancilhon, F., Spyratos, N.: Update semantics of relational views. ACM Trans. Database Syst. 6, 557–575 (1981)
2. Barr, M., Wells, C.: Category theory for computing science, 2nd edn. Prentice-Hall, Englewood Cliffs (1995)
3. Barr, M., Wells, C.: Toposes, Triples and Theories. Grundlehren Math. Wiss. 278 (1985)
4. Bohannon, A., Pierce, B., Vaughan, J.: Relational Lenses: A language for updatable views. In: Proceedings of ACM PODS 2006 (2006)
5. Dayal, U., Bernstein, P.A.: On the correct translation of update operations on relational views. ACM TODS 7, 381–416 (1982)
6. Hegner, S.J.: An order-based theory of updates for closed database views. Annals of Mathematics and Artificial Intelligence 40, 63–125 (2004)
7. Johnson, M., Rosebrugh, R.: View updatability based on the models of a formal specification. In: Oliveira, J.N., Zave, P. (eds.) FME 2001. LNCS, vol. 2021, pp. 534–549. Springer, Heidelberg (2001)
8. Johnson, M., Rosebrugh, R., Wood, R.J.: Entity-relationship-attribute designs and sketches. Theory and Applications of Categories 10, 94–112 (2002)
9. Johnson, M., Rosebrugh, R.: Fibrations and universal view updatability. Theoretical Computer Science (in press, 2007)
10. Lechtenbörger, J.: The impact of the constant complement approach towards view updating. In: Proceedings of ACM PODS 2003, pp. 49–55 (2003)
11. Rosebrugh, R., Fletcher, R., Ranieri, V., Green, K.: EASIK: An EA-Sketch Implementation Kit, `http://www.mta.ca/~rrosebru`
12. Shipman, D.: The functional data model and the data language DAPLEX. ACM Trans. Database Syst. 6, 140–173 (1981)