Fibrations and universal view updatability

Michael Johnson*
Computing Department, Macquarie University NSW 2109, Australia
mike@ics.mq.edu.au

Robert Rosebrugh*
Department of Mathematics and Computer Science,
Mount Allison University, Sackville, NB E4L1G6, Canada
rrosebrugh@mta.ca

Abstract

Maintainability and modifiability of information system software can be enhanced by the provision of comprehensive support for *views*, since view support allows application programs to continue to operate unchanged when the underlying information system is modified. Supporting views depends upon a solution to the *view update problem*. This paper presents a new treatment of view updates for formally specified semantic data models based on the category theoretic sketch data model. The sketch data model has been the basis of a number of successful major information system consultancies. We define view updates by a universal property in models of the formal specification, and explain why this indeed gives a complete and correct treatment of view updatability, including a solution to the view update problem. However, a definition of updatability which is based on models causes some inconvenience in applications, so we prove that in a variety of circumstances updatability is guaranteed independently of the current model. This is done first with a very general criterion, and then for some specific cases relevant to applications. We include some detail about the sketch data model, noting that it involves extensions of algebraic data specification techniques.
Keywords: Database, semantic data model, view update, category, fibration.

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

1 Introduction

Views on database schemata and their updates have been the subject of considerable research, but the implementation of view updates, especially within standards such as the database language SQL, has been largely based on ad hoc and restrictive requirements. One reason for this is the failure to obtain a comprehensive solution to the problem of determining what updates *should* be allowed to a view of a database. Furthermore, the relatively limited use of formal methods (as opposed to semi-formal methodologies) has often resulted in views being defined either informally, or solely in terms of the underlying information system's base schema. Both of these hamper the use of the view mechanism in facilitating program reuse when the underlying information system changes.

It is well known that not all updates to a view of a database can be permitted. View data are derived from an underlying database, and apparently acceptable changes to a view may be prohibited or ambiguous if applied to the underlying database. The *view update problem* is to determine when and how updates of views can be correctly propagated to the underlying database. We will see many examples below, but for a text book treatment the reader is referred to Chapter 10 of Date's text [16].

The authors and coworkers have developed a semantic data model for information systems based on *category theory* and *sketches*. The impetus for this development came from a very large consultancy [11] which compelled the use of formal methods to manage the complexity. Dampney and Johnson [25] first showed how data models based on entity-relationship (ER) diagrams [10] are enhanced by the inclusion of concepts from category theory such as commutative diagrams, finite limits and sums to express constraints and queries. The semantic data model we are using has recently been called the *sketch data model* because it is based upon the category theoretic notion of mixed sketch [5], here specialized for our purposes to the *EA sketch*. The sketch data model has been developed considerably and tested in other consultancies for an Australian department of health [12] and a large corporation [14]. We believe that these successful applications, together with the theory developed here to provide a new approach to views and to clarify the view update problem, provide strong support for this form of semantic data modelling.

In Section 2 we describe a motivating example in some detail, and then define EA sketches and their models and updates. In Section 3, we consider the query language for an EA sketch and use it with the notion of sketch morphism to define "view". This is done in a manner that is designed to permit the widest possible range of data accessible from the underlying database to be structured in the view.

A solution to the view update problem needs to determine when and how view updates can be propagated to the underlying database state. Typically the *when* has been determined by definition—certain view updates are defined to be updatable, and then for those updates

the *how* is given by specifying the propagation of each view update into an update of the underlying database. Unfortunately this approach has led to ad hoc treatments as each discovery of new *how*s leads to an adjustment of the defined *when*s, and naturally we should not expect such solutions to be complete—they really represent a list of those view updates that a system is currently able to propagate. For an example of the evolution of ideas about updatability consult successive editions of Date's text [16]. Instead, in Section 4 we define the *when* and *how* together using a *universal property*, that is we seek a best possible update of the underlying database state which restricts to the given view state update. Our universal criterion for view updatability may be summed up globally with the concepts of *fibration* and *opfibration*. This provides a clear conceptual basis for updatability of views.

The universal property we use is based on the (view) database state. However, view updatability has usually been treated with criteria based on database schemata. We address this in two ways. First, in Section 5 we develop general criteria on views that guarantee universal updatability. They are quite restrictive in the relation the view sketch may have to the underlying sketch. Thus, in Section 6 we also prove that view updatability can be determined independently of the database state in a variety of special cases. Such results considerably simplify the application of the theory.

Preliminary versions of some of the ideas in this article, though not the central definitions and results of Sections 4 and 5, are found in the articles [15] and [29].

2 EA Sketches and Models

This section provides background on the sketch data model and EA sketches. Some familiarity with the language of categories and functors applied to computer science, as found in the books [5], [37] or [46], is assumed.

Before providing the definitions we will need, we give extended consideration to a motivating example derived from a fragment of a health administration model. First we describe the application area, how it might appear in an ER diagram and the resulting relational database schema. Then we describe an EA sketch whose components include the preceding information and whose constraints model the application area more precisely.

The main entities in our example are hospitals and medical practitioners. The latter may be general practitioners (GP's) or specialists. Relationships between them include operations by a practitioner at a hospital, and practice agreements (a type of contract) between practitioners and hospitals. The entities may have attributes such as the name and address of a hospital, or the licence number of a medical practitioner. Relationships may also have attributes such as the type of an operation. An entity-relationship-attribute (ER) diagram capturing some of this information might look like the graph in Figure 1.

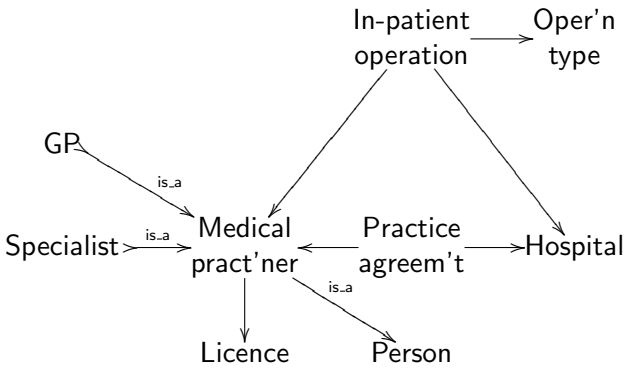


Figure 1: A fragment of an ER diagram

There is no agreed standard for display of nodes and edges in such graphs. We note that by framing them differently, typical ER diagrams would distinguish entity nodes like Medical pract'ner or Hospital from relationship nodes like Practice agreem't, and from attribute nodes like Licence and Oper'n type. ER diagrams often use undirected edges to indicate the entities participating in a relationship and to indicate attributes for an entity. Attributes may be decorated to indicate that they form a key (or unique identifier) for their entity. Decorations on the edges for a relationship may indicate "multiplicity": whether instances of an entity may appear at most once in a relationship. Other decorations may indicate subtyping—"is_a" in ER terminology. It is perhaps worth noting that the ER data model dates to 1976 [10], well before object orientation and UML became fashionable in Computer Science, but they have ideas in common with these later developments.

In the graph above we use directed edges from a relationship to the participating entities, and a directed edge from an entity or relationship to its attributes. This varies from most ER diagram conventions. We have shown only some of the many attributes that might be part of the design. The graph also has three edges with tails labelled is_a to indicate subtyping. For example, a GP is a Medical pract'ner.

In implementations, an ER diagram is translated into a relational database schema: a set of relation schemes (table specifications) and database constraints. An entity determines a relation scheme whose attributes (column headings) must be the attributes of that entity, often specifying a unique identifier attribute. For example, the Medical pract'ner entity determines a relation scheme Medical_practitioner(Licence, ...). Licence (number) is an attribute. It may be specified to uniquely determine a medical practitioner, In a model, any medical practitioner instance has a unique set of values for its other attributes. Such an attribute is called a (primary) key in the database literature. The relationships each determine a

relation scheme with an attribute for each participating entity (or better, for its unique identifier), and for each other attribute. Thus, the Practice agreem't relationship defines a relation schema Practice_agreement(Licence, Hospital_Name). To ensure database integrity, in any instance of the relationship the attributes from participating entities must refer to extant instances of the entities. Thus, they are examples of "foreign keys" in database terms. There are several ways of translating subtyping edges; one way is to include the supertype unique identifier as a subtype attribute which must also be a foreign key, so we might have a relation scheme GP(..., Licence, ...).

There are facts about the modelled environment that an ER diagram misses. For example, we want to require that a Medical pract'ner is either a GP or a Specialist, but not both. An In-patient operation should occur under a Practice agreem't. The agreement might become an attribute of the operation, but we want the practitioner and hospital involved to be determined by the relationship. One might suggest making the agreement a participating entity in the operation relationship, but Practice agreem't is a relationship, not an entity!

The ER data model certainly aids good design: graphical representation is a powerful design tool. However, the brief example above suggests that the ER model may fail to capture quite simple constraints. Modern relational database systems are able to implement constraints like those described above, but it is reasonable to expect that designing the constraints should be part of the data model, rather than hoping that they will be noticed, and implemented, later by application programmers.

We are ready to introduce EA sketches and the sketch data model. An EA sketch is a special case of mixed sketch and thus has four components.

a) An underlying *directed graph* G whose nodes specify entities and attributes (similar to an ER diagram).

b) A set of *commuting diagrams.* A commuting diagram is a pair of paths in G with common source and target. They are used to specify equality of functional constraints.

c) A set of finite *cones*. A cone has three parts: a node v of G called the *vertex*; a graph morphism $\gamma : I \longrightarrow G$ with I finite called the *base*; and, for each node i of I, a *projection* edge $p_i : v \longrightarrow \gamma(i)$ from the vertex to base (codomain) nodes. They are used to specify monic (`is_a`), and also `select`, `project`, `join` constraints.

d) A set of finite *cocones*. Like cones, cocones have a vertex v and base γ, but they have *injection* edges $j_i : \gamma(i) \longrightarrow v$. Only discrete cocones appear in EA sketches. A cocone is *discrete* when the domain graph

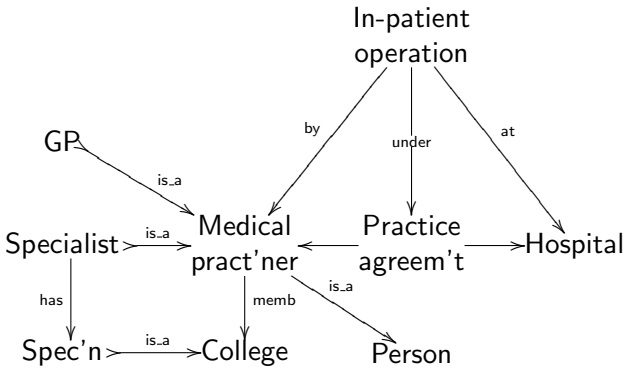


Figure 2: A fragment of a graph for an EA sketch

of its base has no edges. They are used in EA sketches both to define attributes and to specify type hierarchies.

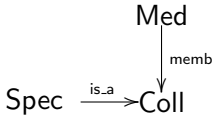
The first component, and some aspects of the third are all that can be expressed with an ER diagram.

Example 1 Figure 2 shows the interesting part of the underlying graph of a small EA sketch $\mathbb{E}$. It is a fragment of the data model for a consultancy [12]. It is similar to the ER diagram in Figure 1, with additional information not expressible there. The other three components of $\mathbb{E}$ are as follows:

b) The set of commuting diagrams of $\mathbb{E}$ contains both triangles and the square.

c) The set of cones contains:

- a cone specified as follows: the vertex is Specialist, the upper left corner of the square; the base is the graph morphism from the following graph I:



which maps the edges eponymously to the right side and the bottom of the square in Figure 2; the projections are: the top edge is_a of the square, the left side edge has of the square, and the common value (since the square commutes) of the diagonal memb∘is_a (an edge we did not draw);

- three further cones (that we have not drawn) ensure that the three arrows indicated $\rightarrowtail$ are modelled by monic (injective) mappings; as we discuss below, not all edges in the base of these cones are shown;
- a special cone with vertex denoted 1 (also not shown) and empty base.

d) The set of discrete cocones contains:

- the cocone with vertex Medical pract'ner and base the vertices Specialist and GP (short for General Practitioner) and
- a number of cocones whose vertices are attributes (not shown, but recall Licence and Oper'n type above).

Briefly, we expand on each of the components in turn, indicating also how they will be modelled in database states.

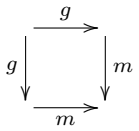
a) The graph may be viewed as a type diagram [44]. The nodes in Figure 2 are all "entities"—the "E" in EA. The three monic arrows (denoted $\rightarrowtail$) indicate subtypes. The other arrows are functions (and can be thought of as methods or, from the database perspective, as integrity constraints). Given an instance of their source type they return an instance of their target type. The names on nodes and arrows have no formal significance but do indicate the real world semantics being captured in the specification and will be used from now on in our discussion.

b) The *commutativity* of the two triangles represents a typical real-world constraint: Every in-patient operation conducted at a particular hospital by a particular medical practitioner must take place under a practice agreement (a type of contract) between that hospital and that practitioner. This constraint is expressed by the requirement that the left hand triangle is a commutative diagram. If, instead, the left hand triangle were not required to commute, then it would still be the case

that every operation took place under an agreement, but Dr. X could operate under Dr. Y's practice agreement. In many information models, situations like this do not even include the arrow labelled under. Such models store the contractual information, but do not specify the constraint—it is expected to be added at implementation time. This example shows that information modelling is *not* usually a complete formal specification technique.

c) The cone involving the square in Figure 2 ensures that in models the square will be a pullback (in database terminology this means it is a `join` [1]). This constraint ensures that the specialists are precisely those medical practitioners who are members of a college which occurs in the subtype Specialization (denoted Spec'n in Figure 2). Pullbacks can also be used to specify further subtypes, for example, the medical practitioners with a specific specialization, say obstetricians.

   Subtype inclusion arrows, and other arrows that are required to be monic in models, are specified using cones like the one just considered. Indeed, to require that a square



   defines a cone as above is equivalent to requiring that in models the square becomes a pullback and m is modelled by a monic. In Figure 2 we drew only the monic `is_a` edges to stand for such cones.

d) The cocone with vertex Medical practitioner ensures that in models the collection of medical practitioners is the disjoint union (categorically, the sum or coproduct) of the collection of specialists and the collection of GPs.

   As is usual practice, we did not draw the attributes—the "A" in EA—in Figure 2, but they are definitely a part of the EA sketch. Attributes are (usually large) fixed, value sets. Examples in this case are the name and the address of a person, the validity period of a practice agreement, the classification of a hospital, the type of an operation, the licence number of a medical practitioner and so on. An attribute is the vertex of a cocone whose base is a finite discrete graph, whose

nodes are all specified by the special node 1, and whose injection edges from 1 are called *elements*. In models, an attribute's value is exactly the sum of its elements. Formally the cocones that define attributes are expressed using the underlying graph. In practice they are usually listed separately in a data dictionary.

Examples of attributes as types include `integer` (with specified bounding value), `string` (of specified maximum length), and `date` etc. In models these types arise as a finite sum of 1 ($1+1+\ldots+1$). Such sums have the important technical property of having constants, the injections, corresponding to every one of their elements. These are usually the only constants that occur in modelling real world data.

We now proceed with the formal definitions required for EA sketches and their model categories. Up to Definition 6 these are standard ([5]) and are included only to establish notation.

Definition 2 *A* cone $C = (v, B, p)$ *in a graph* G *consists of a node* v *of* G *(the* vertex *of* C*), a graph morphism* $B : I \longrightarrow G$ *(the* base diagram *of* C*), and, for each node* i *in* I*, an edge* $p_i : v \longrightarrow Bi$*.* Cocones *are dual (that is we reverse all the edges of* G *which occur in the definition, so the new definition is the same except that the last phrase requires edges* $j_i : Bi \longrightarrow v$*). The edges* p_i *in a cone (respectively* j_i *in a cocone) are called* projections *(respectively* coprojections *or* injections*).*

The general definition of a *mixed* sketch (i.e. a sketch with both cones and cocones) follows in the next paragraph. Sketches with a variety of restrictions on the graph, cones or cocones have been considered. For example, if the cones are finite and there are no cocones at all we have the widely studied *finite limit* sketches. Our EA sketches have the restrictions noted in Definition 7.

Definition 3 *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*.*

Every category $\mathbf{A}$ has an underlying sketch. For this sketch the graph G is the underlying graph of the category, $\mathbf{D}$ is the set of all commutative diagrams, and $\mathcal{L}$ (respectively $\mathcal{C}$) is the set of all limit cones (respectively colimit cocones) in the category. Note that limit and colimit cones in a category $\mathbf{A}$ are required to be both *commutative* and *universal*. Commutativity means that for any edge $i \xrightarrow{e} i'$ in I, we have $p_{i'} = B(e)p_i$ (and dually for cocones). Commutativity is not required of the cones in a sketch—after all G is merely a graph. Universality of a cone C in a category $\mathbf{A}$ means that there is a natural bijection

between commutative cones C' in $\mathbf{A}$ with vertex v' say, and base B, and arrows in $\mathbf{A}$ from v' to v. Pullbacks, equalizers and products examples of limit cones. Pushouts, coequalizers and sums are colimits. The underlying sketch of a category often has, for example, infinitely many commutative diagrams or cones. However, one advantage of sketches is that we can frequently use a sketch with finite components to give a presentation of an infinite category, just as we do for groups.

Definition 4 *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 $h : \mathbb{E} \longrightarrow \mathbb{E}'$ *is a graph morphism* $G \longrightarrow G'$ *which carries, by composition, 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* $\mathcal{C}'$*.*

For example, a functor is the same thing as a sketch morphism between the underlying sketches of its domain and codomain categories which also preserves limits (takes limit cones to limit cones) and colimits. The definition of functor entails that commutative diagrams are preserved. More important for us are the *models* of a sketch:

Definition 5 *A* model M *of a sketch* $\mathbb{E}$ *in a category* $\mathbf{S}$ *is a sketch morphism from* $\mathbb{E}$ *to the underlying sketch of the category* $\mathbf{S}$*.*

Equivalently, we can describe models using functors as follows. Let $C(\mathbb{E})$ be the category generated by the graph G, subject to the equations given by the commutative diagrams in $\mathbf{D}$. Thus $C(\mathbb{E})$ has the nodes of G as objects and equivalence classes of paths in G as arrows. There is evidently an inclusion (of graphs) $G \longrightarrow C(\mathbb{E})$ so we will continue to refer to nodes of G as objects, edges of G as arrows and so on. A model of $\mathbb{E}$ in $\mathbf{S}$ is exactly the same thing as a functor $M : C(\mathbb{E}) \longrightarrow \mathbf{S}$ such that the cones in $\mathcal{L}$ and cocones in $\mathcal{C}$ are sent to limit cones and colimit cocones in $\mathbf{S}$. Thus, the models of $\mathbb{E}$ in $\mathbf{S}$ are some of the objects of the category of functors from $C(\mathbb{E})$ to $\mathbf{S}$, denoted $[C(\mathbb{E}), \mathbf{S}]$.

Definition 6 *If* M *and* M' *are models of* $\mathbb{E}$ *(viewed as functors) a* homomorphism $\phi : M \longrightarrow M'$ *is a natural transformation from* M *to* M'*.*

Models and their homomorphisms are the objects and arrows of *the category of models* of $\mathbb{E}$ in $\mathbf{S}$ denoted by $\mathrm{Mod}(\mathbb{E}, \mathbf{S})$, namely the full subcategory of $[C(\mathbb{E}), \mathbf{S}]$ determined by the models.

We speak of (limit-class, colimit-class)-sketches when $\mathcal{L}$ and $\mathcal{C}$ are required to contain (co)cones only from the specified (co)limit classes. For example, (finite-discrete, $\emptyset$)-sketches correspond to (multi-sorted) algebraic theories—there are only finite discrete (product) cones in $\mathcal{L}$ and no cocones in $\mathcal{C}$ at all. The model functors M preserve only the specified finite

discrete cones (and of course any commutative diagrams), so the discrete cones are sent to product cones. As is well known, algebraic theories define precisely the situation in which category theory applies to the algebraic specification of data structures [19]. Our EA sketches have finite cones and finite discrete (i.e. sum) cocones.

Definition 7 *An* EA sketch $\mathbb{E} = (G, \mathbf{D}, \mathcal{L}, \mathcal{C})$ *is a (finite limit, finite discrete cocone)-sketch such that there is a specified cone with empty base in* $\mathcal{L}$*. Its vertex will be called* 1*. Arrows 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*.*

Using Figure 2 as (the main part of the) underlying graph and taking together the commutativity, cones, cocones and attribute information as described in Example 1 we obtain an EA sketch. According to Definition 7, attributes are finite sums of elements. Thus, for models in $\mathbf{set}_0$, the category of finite sets, attributes will be modelled as finite sets, for example strings of some bounded length, rather than infinite sets, for example all strings.

A brief comment on the name: As the reader will have expected, EA stands for Entity-Attribute. Relationships in an ER diagram can be replaced by their *tabulations*, that is the span of arrows from the relationship to its entities. For example, the ER diagram in Figure 1 had Practice agreement as a relationship from Medical practitioner to Hospital. It may initially appear that we have merely resurrected the functional data model [42]. Of course, the sketch data model does use functions in its modelling, as the next definition emphasizes. The difference is the integral role played by encoding constraints using commutative diagrams, finite limits and sums.

To specify a database state for an EA sketch is to give a collection of finite sets and functions among them as specified by the nodes and the edges of the graph. These are required to satisfy the constraints.

For example, the finite set corresponding to In-patient operation in Figure 2 is the collection of all operations currently stored in the database. The satisfaction of the three monic constraints means that arrows denoted $\rightarrowtail$ in Figure 2 must be realized as injective functions, and so on.

Our main purpose is to study view updatability. To do so we need to have a clear definition of database states and updates. We will deal with insert and delete updates separately here and they are simply defined.

Definition 8 *A* database state *D for an EA sketch* $\mathbb{E}$ *is a model of* $\mathbb{E}$ *in* $\mathbf{set}_0$*, the category of finite sets. The* category of database states of $\mathbb{E}$ *is the category of models* $\mathrm{Mod}(\mathbb{E}, \mathbf{set}_0)$ *of* $\mathbb{E}$ *in* $\mathbf{set}_0$*, abbreviated below as* $\mathrm{Mod}(\mathbb{E})$*. An* insert update *(respectively* delete update*) for a database state D is a monomorphism* $D \rightarrowtail D'$ *(respectively* $D' \rightarrowtail D$*) in the category* $\mathrm{Mod}(\mathbb{E})$ *of models of* $\mathbb{E}$*.*

Since morphisms of database states are natural transformations, to satisfy the definition of update it is sufficient that at each node of the underlying graph the component of the morphism is injective. Thus, the definition implements the intuitive idea that an insert update adds new values at entities. Indeed, an insert update may add values at several entities. For the case of an update that involves adding instances at only one entity, we have published algorithms to accomplish this for certain classes of EA sketches [27],[28].

In this article we deal only with delete and insert updates. In practice this is a very small limitation—all common real-world updates can be expressed as delete-insert pairs. In precise terms, for an EA sketch in which no entity has an element (in the sense of Definition 7), all updates can be expressed as a delete, then an insert. Even when entities do have such elements (constants), most updates can still be expressed as delete-insert pairs. An example of an update which cannot be so expressed, one which must be carried out *in situ* is as follows. Consider the modification of an attribute value for a specified instance of an entity which is constrained so that, in any model, the value of the entity is *always* required to have ten (say) instances. Since there can be no deletes from or inserts into that entity, updating via a delete-insert pair is not possible here.

The sketch data model could be viewed as an extension of the entity-relationship-attribute (ER) model. Indeed, any ER diagram will generate the graph for an EA sketch (the details are dealt with in the article [25]). However, the sketch arising from the ER diagram fails to include commutativity requirements, finite limit constraints (except for the $\mathsf{is\_a}$ monics), or sum constraints (except for the attributes). As Example 1 indicates, such constraints add significantly to the expressivity of an EA sketch.

3 Queries and Views

We begin this section by discussing the query language that is implicitly defined by an EA sketch.

To each EA sketch $\mathbb{E}$ there is a category called the *theory* of the sketch. We denote this category $Q\mathbb{E}$. The theory $Q\mathbb{E}$ is, roughly speaking, the category generated from the sketch $\mathbb{E}$ subject to the commutative diagrams in $\mathbf{D}$ by closing it under the operations of finite limit and finite sum subject to the (co)cones in $\mathcal{L}$ and $\mathcal{C}$. Thus the theory of the sketch in Example 1 has infinitely many nodes in its underlying graph including in particular the product of any two nodes shown Figure 2, the sum of any two nodes shown in Figure 2, pullbacks of any cospan shown in Figure 2, etc, and then the products, sums and pullbacks of all these products, sums and pullbacks, and so on. For details and a precise construction see Barr and Wells [6], Section 8.2. Thus, $Q\mathbb{E}$ has all finite limits and finite sums. The inclusion of the underlying graph of $\mathbb{E}$ in that of $Q\mathbb{E}$ is actually a model $\mathbb{E} \longrightarrow Q\mathbb{E}$ of $\mathbb{E}$

in $Q\mathbb{E}$. Moreover, it is the *initial model* essentially because $Q\mathbb{E}$ contains exactly $\mathbb{E}$ and expressions for finite limits and finite sums generated by the specifications of $\mathbb{E}$. Indeed, if a category $\mathbf{S}$ has finite limits and finite sums (as $\mathbf{set}_0$ does) then every model M of $\mathbb{E}$ in $\mathbf{S}$ extends essentially uniquely to a functor $QM : Q\mathbb{E} \longrightarrow \mathbf{S}$ which is moreover a model of (the underlying sketch of) $Q\mathbb{E}$ in $\mathbf{S}$. Thus, Mod($\mathbb{E}$) is equivalent as a category to Mod($Q\mathbb{E}$) (the extension works on morphisms of the model categories as well). We explore the meaning of this for the sketch data model next.

The extra semantic power of the sketch data model comes from the non-graph components: As seen in Example 1, $\mathbf{D}$ can be used to specify constraints defined by paths in the graph, and $\mathcal{L}$ and $\mathcal{C}$ can be used to specify constraints that may also be viewed as query results. The theory of an EA sketch includes objects representing *all* queries expressible as finite limits or finite sums in the objects and arrows of the base graph, and then these query results can be used to specify further queries and constraints, and so on.

Which standard database queries are expressible by objects of $Q\mathbb{E}$? The answer is that *all* basic `select`, `project`, `join` and (disjoint) `union` queries are included. That pullbacks express `join`s has been mentioned above. Finite sums give disjoint unions. Since finite products are examples of finite limits, the results of `project` queries arise from their projections. Finally, basic `select` queries arise using *equalizers*. For example, to determine the In-patient operations performed by a particular Medical practitioner Jones, say, we form the equalizer of the arrow by and the composite arrow

$$\text{In-patient operation} \longrightarrow 1 \xrightarrow{\text{Jones}} \text{Medical practitioner}$$

The resulting equalizer gives the result for the (pseudo-SQL) query

`select * from` In-patient operation `where` by $=$ "Jones"

Since the query category $Q\mathbb{E}$ is closed under finite limits and finite sums, all such queries as well as all queries containing such sub-queries are expressible by objects of $Q\mathbb{E}$. As noted above, the objects of $Q\mathbb{E}$ express all and exactly the queries built from iteration of finite limits and sums subject to the defining components of $\mathbb{E}$. Thus a single object can express a query of any complexity.

It is not surprising that the provision of a much richer language for specifying constraints is of major benefit in information modelling. Indeed, it has been the basis of the successful consultancies which have used the sketch data model as a formal information modelling technique. Despite the richer specification language provided by the sketch data model, its query language, as noted above, is quite similar in power to SQL.

A *view* (also known as "external view" [16] or "subscheme" [45]) allows a user of an information system to manipulate data which are part of, or are derived from, an underlying

database. The example medical informatics sketch (Figure 2) might represent a view of a large health administration database. In turn it might provide views to an epidemiologist who only needs to deal with the two triangles, and with their associated attributes; or to an administrator of a College of Surgeons who needs to deal with data in the inverse image of that college, and not with any of the data associated only with other colleges.

In relational systems, the formal definition of a view has generally been limited to the result of a single query (a virtual table), for example in the SQL `create view`. Moreover, updatability of views has generally been implemented in very restrictive fashion to avoid the difficulties of the view update problem. For example, allowable updatable views might be restricted to be just certain row and column subsets of a relational database. Join views are usually considered to be not updatable (see [45], Section 6.7).

However, we seek to define views which can be derived more generally from the underlying database. We argue that views ought to be able to be structured in any way consistent with the data model in use. Furthermore, the updatability of view states ought to be defined as widely as possible consistent with unique and correct updating of the underlying database state.

This motivates a definition of view for the sketch data model which supports the generality just described. It uses the theory $Q\mathbb{E}$ constructed from an EA sketch $\mathbb{E}$.

Definition 9 *A* view *of an EA sketch* $\mathbb{E}$ *is an EA sketch* $\mathbb{V}$ *together with a sketch morphism* $V : \mathbb{V} \longrightarrow Q\mathbb{E}$.

Thus a view of $\mathbb{E}$ includes a new EA sketch $\mathbb{V}$ with the entities of $\mathbb{V}$ interpreted via the sketch morphism V as query results in the original EA sketch $\mathbb{E}$.

For example, Example 1 may be considered as a view of a much larger EA sketch. Here the graph part of the sketch morphism might be just an inclusion in the graph part of the larger sketch. Other examples of views, including views which depend on entities in $Q\mathbb{E}$ but not in $\mathbb{E}$, are found in following sections.

A database state D for an EA sketch $\mathbb{E}$ is first of all a sketch morphism $D : \mathbb{E} \longrightarrow \mathbf{set}_0$. However as noted above, it can also be considered (via its essentially unique extension) as a model $D : Q\mathbb{E} \longrightarrow \mathbf{set}_0$. The composite of this model with a view V, is a composite DV of sketch morphisms and so is a sketch morphism which is moreover a database state for $\mathbb{V}$: the V-view of D. This operation of *composing with* V is usually written as V^*. Thus we denote DV by V^*D. Furthermore, V^* is a functor (called the *substitution functor*)

$$V^* : \mathrm{Mod}(\mathbb{E}) \longrightarrow \mathrm{Mod}(\mathbb{V})$$

(where we are eliding the equivalence of $\mathrm{Mod}(\mathbb{E})$ and $\mathrm{Mod}(Q\mathbb{E})$) so for any morphism of database states $\alpha : D \longrightarrow D'$ there is a morphism $V^*\alpha : V^*D \longrightarrow V^*D'$.

Following usual practice we will often refer to a database state of the form V^*D as a "view" of D. Context will determine whether "view" refers to such a state, or to the sketch morphism V. If there may be ambiguity, or for emphasis, we will sometimes refer to V^*D as a *view state*, to $\mathbb{V}$ as the *view sketch* and to V as the *view schema*.

With this definition, views can be composed. For example, in the view $V : \mathbb{V} \longrightarrow Q\mathbb{E}$ of an EA sketch $\mathbb{E}$, $\mathbb{V}$ is itself an EA sketch on which we might define a view $W : \mathbb{W} \longrightarrow Q\mathbb{V}$. The operation of forming the theory of an EA sketch is well behaved, so we have $QV : Q\mathbb{V} \longrightarrow QQ\mathbb{E}$, but since $Q\mathbb{E}$ was formed by closing under appropriate operations, it is already closed under the operations and so $QQ\mathbb{E} \simeq Q\mathbb{E}$. Thus the composite

$$\mathbb{W} \longrightarrow Q\mathbb{V} \longrightarrow QQ\mathbb{E} \simeq Q\mathbb{E}$$

is a sketch morphism. It defines the composite of the views W and V as a view of $\mathbb{E}$. This explains how the epidemiologist above has a (composed) view of the full health administration database.

4 Universal updatability

We have defined *view* above so as to ensure that views have the widest possible applicability. A view has an EA sketch as its domain, and can include any structural form expressible in the sketch data model. The sketch morphism V defining the view schema guarantees that the constraints on view states imposed by the diagrams, limits and colimits in the view sketch are compatible with the constraints on the underlying database. The fact that V takes values in $Q\mathbb{E}$ permits any query from the underlying database to appear in the view sketch.

The *view update problem* is to determine under what circumstances an update to a view state can be propagated to an update of the underlying database, and how that propagation should take place. The source of the problem is that some updates to a view state may have no, or no unique, propagated update. That is, an insert or a delete which is perfectly valid for a view state itself, may not arise (under V^*) from a unique update to a state of the underlying database, or there may be no such state of the underlying database at all. For example, a college administrator might wish to alter the medical practitioner attribute values for a member of the college, but even though such administrators can see the practice agreements for members of their college, they cannot insert a new practice agreement for a member because they cannot see details about hospitals, and every practice agreement must specify a hospital.

In the sketch data model, view updates can fail in either of two ways:

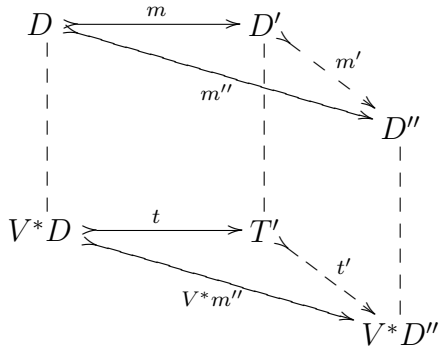
1. There may be no state of the database which would yield the updated view. This usually occurs because the update to the view state, when carried to the underlying database state, would result in a proscribed situation. For example, a view schema might include the product of two entities, but only one of the factors. For a view state, inserting or deleting an instance from the value of the state at the product entity seems straightforward; after all, in the view schema it looks like an unconstrained entity with an arrow to another entity (the projection to the factor that is in the view). But in the underlying database the resulting value of the state at the product entity might be impossible. Indeed, the number of instances in the value at the product entity and the number in the value at the factor entity might have become coprime.

2. There may be many states of the database which would yield the updated view and among these no canonical choice. The simplest example of this occurring is when a view schema includes an entity, but not one of its attributes. Inserting into the entity seems straightforward, but in the underlying database there is no way to know what value the new instance should have on the invisible attribute: there are many choices, but no canonical way to choose among them.

Since a view state is itself a database state, we may (subject to the constraints of the view sketch) insert items in or delete items from a view state. Intuitively, a specified insert/delete in the view is propagatable if there is a unique "minimal" insert/delete on the underlying database whose restriction to the view (via V^*) is the given view insert/delete. Thus we have:

Definition 10 *Let $V : \mathbb{V} \longrightarrow Q\mathbb{E}$ be a view of $\mathbb{E}$. Suppose that D is a database state for $\mathbb{E}$. Let $T = V^*D$ and $t : T \rightarrowtail T'$ be a database state monomorphism (so that t is an insert update of T). The insert t is* propagatable *if there exists an insert update (in* $\mathrm{Mod}(\mathbb{E})$*) $m : D \rightarrowtail D'$ with the following property: for any database state D'' and insert update $m'' : D \rightarrowtail D''$ such that $V^*m'' = t't$ for some $t' : T' \rightarrowtail T''$, there is a unique insert*

$m' : D' \rightarrowtail D''$ *such that* $V^*m' = t'$ *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* T *is propagatable, we say that the view state* T *is* insert updatable.

The definitions of *propagatable delete* and *delete updatable* view state are dual. It should be remarked immediately that, though propagatability only requires the *existence* of the insert update m, if there is any other $n : D \rightarrowtail E'$ satisfying the same requirements as m then there is an isomorphism of models $i : D' \longrightarrow E'$ such that $im = n$, so m is *essentially unique*.

A view schema may be such that all of its view states are insert updatable. For example, a view schema that consisted only of (the inclusion of) the node Hospital (and its attributes) in Figure 2 would have this property (see Proposition 21 below). On the other hand, a view whose only node is Medical practitioner has no propagatable inserts since it is a sum node (see Proposition 26).

There is a technical point to recall about the preceding definition. That is that we are silently using the fact that Mod($\mathbb{E}$) and Mod($Q\mathbb{E}$) are equivalent categories when we write above, for example $T = V^*D$. It should also now be clear just what we meant by "minimal" before Definition 10. Referring to the diagram above, D' has the property that any other D'' which has the appropriate restriction to the view is itself "larger". Another way of saying the same thing is that D' is *initial* among those D'' containing D whose image under V^* contains T'. A propagatable delete dually requires a D' that is maximal (or *terminal*) among those which have the appropriate restriction.

Our definition of propagatability for insertion and deletion (view updatability) is dependent on the view state V^*D (the model of the view sketch) which is being updated. The definition specifies when an insert or delete of a view state is propagatable, rather than for which view schemata inserts and deletes can always be propagated. This is similar to the recent opinion of C. J. Date referring to his proposal for defining view updatability ([16], p 303, Date's emphasis):

> in fact, [this proposal] treats *all* views as potentially updatable,
> barring integrity constraint violations.

Nevertheless, we can sometimes prove for a given view schema that all database states are insert or delete updatable. Such results are important for designers who wish to design views that will always be updatable. Section 6 provides some analysis of this issue.

By defining updatability in terms of models we obtain the broadest reasonable class of updatable view inserts and deletes. Whenever there is a canonical (initial or terminal) state of the underlying database among those models that could achieve the update of the view state, we say that the view state is updatable. The only invalid view state updates are those which are in fact impossible to achieve in the underlying database, or those which could be derived from multiple non-isomorphic minimal (or maximal) models of the underlying database.

Definition 11 *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* $E \rightarrowtail A_E$ *from the entity to the specified attribute.*

This is essentially the requirement of *entity integrity*, for it expresses the presence of primary keys.

In the case of a keyed EA sketch, all morphisms between database states are actually monomorphisms ([33], Proposition 4.7) which means precisely that all of the component mappings are injective. Even when the sketch is not keyed we may consider the category whose objects are all models, but whose arrows are restricted to be monomorphisms. For a sketch $\mathbb{E}$, we denote this category by $\mathrm{Mon}(\mathbb{E})$. When the effect of V^* on a monic morphism of models provides another monic morphism of models we have a restriction $V^* : \mathrm{Mon}(\mathbb{E}) \longrightarrow \mathrm{Mon}(\mathbb{V})$. It is often the case that monomorphic components of an arrow in $\mathrm{Mod}(\mathbb{E})$ are sent by V^* to monomorphic components in $\mathrm{Mod}(\mathbb{V})$ (and always so for the components of entities in $\mathbb{V}$ that V sends into $\mathbb{E}$ rather than properly into $Q\mathbb{E}$). The notation V^* is used both for the functor named above and for its restriction (when defined) to $\mathrm{Mon}(\mathbb{E})$.

The situation in which *all updates of a view are propagatable* is clearly a property of the functor V^*, namely that it is a *left* and *right fibration* in the sense of the following definitions.

Definition 12 *[8] Let* $P : \mathbf{E} \longrightarrow \mathbf{B}$ *be a functor,* E *be an object of* $\mathbf{E}$*, and* $b : B \longrightarrow PE$ *be an arrow of* $\mathbf{B}$*. An arrow* $a : E' \longrightarrow E$ *in* $\mathbf{E}$ *is called a* cartesian arrow *for* b *if* $P(a) = b$ *and for any arrow* $a' : E'' \longrightarrow E$ *such that* $P(a') = bb'$ *for some* $b' : P(E'') \longrightarrow B$*, there is a unique arrow* $k : E'' \longrightarrow E'$ *satisfying* $ak = a'$ *and* $P(k) = b'$*. An* opcartesian arrow *for an arrow* $d : PE \longrightarrow B$ *is an arrow* $c : E \longrightarrow E''$ *with the property dual to that for cartesian arrows.*

Thus, provided that we consider only monomorphisms (in the diagram of Definition 10), our definition of propagatable insert is precisely having an opcartesian arrow (and a propagatable delete has cartesian arrow).

Definition 13 *A functor $P : \mathbf{E} \longrightarrow \mathbf{B}$ is called a* left fibration *(or just* fibration*) if there is a cartesian arrow for every $b : B \longrightarrow PE$. It is a* right fibration *(or* opfibration*) if there is an opcartesian arrow for every $d : PE \longrightarrow B$.*

The following proposition follows immediately from the definition just given and from our definition of propagatable updates. Nevertheless, it expresses a significant point of this article: there is a *universal criterion for view updatability*, namely that the substitution functor V^* is a left and right fibration.

Proposition 14 *Let $\mathbb{E}$ and $\mathbb{V}$ be EA sketches and $V : \mathbb{V} \longrightarrow Q\mathbb{E}$ a view of $\mathbb{E}$. V^* is a left (respectively, right) fibration (with domain* $\mathrm{Mod}(\mathbb{E})$ *in the keyed case and* $\mathrm{Mon}(\mathbb{E})$ *otherwise) if and only if every database state T in* $\mathrm{Mod}(\mathbb{V})$ *is delete (respectively, insert) updatable.* ∎

5 Cofibrations and updatability

In this section we develop general criteria for a sketch morphism $V : \mathbb{V} \longrightarrow Q\mathbb{E}$ that guarantee that V^* is a right or left fibration. The criteria are based on the notion of *cofibration* for categories and functors developed by Street [43]. The results following hold for general mixed sketches and models in **set**, so for this section we abbreviate $\mathrm{Mod}(\mathbb{E}, \mathbf{set})$ to $\mathrm{Mod}(\mathbb{E})$. The results specialize to EA sketches and their models without change.

We use the standard notation $\mathbf{E}(A, B)$ for the set of morphisms from A to B in the category $\mathbf{E}$. Suppose that $I : \mathbf{A} \longrightarrow \mathbf{E}$ is a fully faithful functor with image closed under isomorphism and that for any object E of $\mathbf{E}$ not in the image of I it is the case that $\mathbf{E}(E, IA)$ is empty for every A in $\mathbf{A}$, then I is called a *left cofibration*. A *right cofibration* $J : \mathbf{B} \longrightarrow \mathbf{E}$ is a fully faithful functor with image closed under isomorphism such that for any object E of $\mathbf{E}$ not in the image of $J : \mathbf{B} \longrightarrow \mathbf{E}$ it is the case that $\mathbf{E}(IB, E)$ is empty for every B in $\mathbf{B}$.

The following result is well-known and we just sketch a proof below. For categories $\mathbf{A}$ and $\mathbf{B}$, the category of functors from $\mathbf{A}$ to $\mathbf{B}$ is denoted by $\mathbf{B}^{\mathbf{A}}$.

Lemma 15 *[43] If $I : \mathbf{A} \longrightarrow \mathbf{E}$ is a left (respectively right) cofibration, then for any category $\mathbf{X}$ the functor $I^* : \mathbf{X}^{\mathbf{E}} \longrightarrow \mathbf{X}^{\mathbf{A}}$ defined by $I^*F = FI$ is a left (respectively right) fibration.*

The interested reader will have no difficulty constructing the required (op)cartesian arrows. We provide some hints in the right fibration case. Suppose that $F : \mathbf{E} \longrightarrow \mathbf{X}$ and $\beta : I^*F \longrightarrow K$ is a morphism in $\mathbf{X}^{\mathbf{A}}$. We need an opcartesian arrow $\beta_* : F \longrightarrow F\beta$ in $\mathbf{X}^{\mathbf{E}}$. The functor $F\beta$ (an object of $\mathbf{X}^{\mathbf{E}}$) is constructed using identities wherever it is not already determined by the value of K on objects of $\mathbf{A}$. The (op)cartesian arrow β_* is then the natural transformation with components given by β on the image of $\mathbf{A}$ in $\mathbf{E}$ and identity components elsewhere.

Our interest is to extend the results in Lemma 15 to substitution functors (those called V^* above) between categories of models of EA sketches and so we begin with a sketch morphism V. In this case both the domain and codomain of the fibration are full subcategories of the functor category $\mathbf{X}^{\mathbf{E}}$ where $\mathbf{E} = C(\mathbb{E})$ and $\mathbf{X} = \mathbf{set}_0$. We will need an appropriate restriction on our sketch morphisms:

Definition 16 *A sketch morphism* $V : \mathbb{V} = (G_V, \mathbf{D}_V, \mathcal{L}_V, \mathcal{C}_V) \longrightarrow \mathbb{E} = (G_E, \mathbf{D}_E, \mathcal{L}_E, \mathcal{C}_E)$ *is a* sketch embedding *if*

- *the graph morphism* V *is injective on objects and edges;*
- *every edge in* G_E *between nodes in the image of* V *lies in the image of* V*;*
- *every commutative diagram in* $\mathbf{D}_E$ *lying in the image of* V *is the image of a commutative diagram in* $\mathbf{D}_V$*;*
- *every (co)cone in* $\mathcal{L}_E$ *(respectively* $\mathcal{C}_E$*) lying in the image of* V *is the image of a (co)cone in* $\mathcal{L}_V$ *(respectively* $\mathcal{C}_V$*).*

Remark 17 As a consequence, the functor induced by V from $C(\mathbb{V})$ to $C(\mathbb{E})$ is a full embedding.

Definition 18 *A sketch embedding* $V : \mathbb{V} \longrightarrow \mathbb{E}$ *with the property that any (co)cone in* $\mathcal{L}_E$ *(respectively* $\mathcal{C}_E$*) with a node (base or vertex) in the image of* G_V *lies entirely in the image of* G_V *is called a* sketch right cofibration *if there is no edge in* G_E *from a node in the image of* V *to a node not in the image of* V*. Such a* V *is a* sketch left cofibration *if there is no edge in* G_E *from a node not in the image of* V *to a node in the image of* V*.*

The condition on (co)cones guarantees that limits and colimits do not interfere with the constructions used. The remaining conditions are analogous to those defining cofibrations.

Proposition 19 *If* V *is a sketch left (respectively right) cofibration then* $V^* : \mathrm{Mod}(\mathbb{E}) \longrightarrow \mathrm{Mod}(\mathbb{V})$ *is a left (respectively right) fibration.*

Proof. Let M be in Mod($\mathbb{E}$). Recall that we also denote by M the functor from $C(\mathbb{E})$ to **set**.

We consider the case of V a right cofibration. Let $\beta : V^*M \longrightarrow N$ be a homomorphism in Mod($\mathbb{V}$). Construct a *functor* $M\beta : \mathbb{V} \longrightarrow$ **set** and a natural transformation $\beta_* : M \longrightarrow M\beta$ by the construction suggested after Lemma 15.

We claim that $M\beta$ is in Mod($\mathbb{E}$). To see this, since we already know that $M\beta$ is a functor, we have only to verify that $M\beta$ sends any cone in $\mathcal{L}$, respectively cocone in $\mathcal{C}$ to a limit cone, respectively colimit cocone, in **set**. We consider only cones; cocones are similar. Let $C = (v_C, I_C : B_C \longrightarrow G_E)$ be a cone in $\mathcal{L}_E$. By hypothesis, if v_C or any node in the image of I_C is in the image of V then all of C is in the image of V. If this is the case, then by definition of $M\beta$, the image of the cone C is a limit cone in **set** since $M\beta$ is defined by N on all of C, and N is a model by hypothesis. Similarly, if C has no nodes from the image of V then $M\beta$ is defined entirely by M which is a model on all of C, so the image of C is a limit cone.

We conclude that $M\beta$ is in Mod($\mathbb{E}$) as required.

To complete the proof we have to show that $\beta_* : M \longrightarrow M\beta$ satisfies the universal property. By its definition β_* certainly does this with respect to functors that are not necessarily models. Now suppose we have homomorphisms $\gamma : N \longrightarrow V^*P$ in Mod($\mathbb{V}$) and $\delta : M \longrightarrow P$ in Mod($\mathbb{E}$) such that $V^*\delta = \gamma\beta$. Because Mod($\mathbb{E}$) is a full subcategory of the functor category, the natural transformation $\delta_* : M\beta \longrightarrow P$ constructed as required for Lemma 15 is certainly a homomorphism. It is moreover the unique such with the required properties. ∎

Notice that in order to use this proposition to ensure both insert and delete updatability for a view $V : \mathbb{V} \longrightarrow Q\mathbb{E}$ we must have that $C(V) : C(\mathbb{V}) \longrightarrow C(Q\mathbb{E})$ is fully faithful and there are no arrows from the image of $C(V)$ to or from the rest of $C(Q\mathbb{E})$. In the next section we consider some more particular situations where this restriction may be relaxed. Furthermore, the conditions in the Proposition are not necessary. Let G_E have only two entities E_1 and E_2, one edge $e : E_1 \longrightarrow E_2$, and no (co)cones. For the view V given by the inclusion of the entity E_2, it is well known that V^* is both a left and right fibration, but V clearly violates the second condition to be a sketch left fibration.

6 Updatability for schemata

In this section we provide some more particular criteria for updatability of views. In some cases they will establish that V^* is a left and right fibration, while others are more limited. Notice that *only values of entities may be updated* since the value 1 and the values of attributes

(as sums of 1) are essentially fixed in any model. By an abuse of language, we will say that an insert update $T \rightarrowtail T'$ is "into (an entity) W" when $T(V) = T'(V)$ for all entities $V \neq W$, and similarly for deletes "from W". Thus an insert into or delete from W changes the database state's value only at W—the values in the model of other entities and attributes remain unchanged.

Definition 20 *A view $V : \mathbb{V} \longrightarrow Q\mathbb{E}$ is called* insert *(respectively* delete*)* updatable at an entity W *in* $\mathbb{V}$ *when all inserts (respectively deletes) into (respectively from) W are propagatable, independently of the database state.*

We establish insert or delete updatability at W in $\mathbb{V}$ (sometimes loosely just called *updatability*) for a variety of circumstances. As well as being technically useful in applications, these results help to show that the definitions above correspond well to our intuitions about what should and should not be updatable. In most cases we will deal in detail with the insert case as it is the more interesting and slightly harder case.

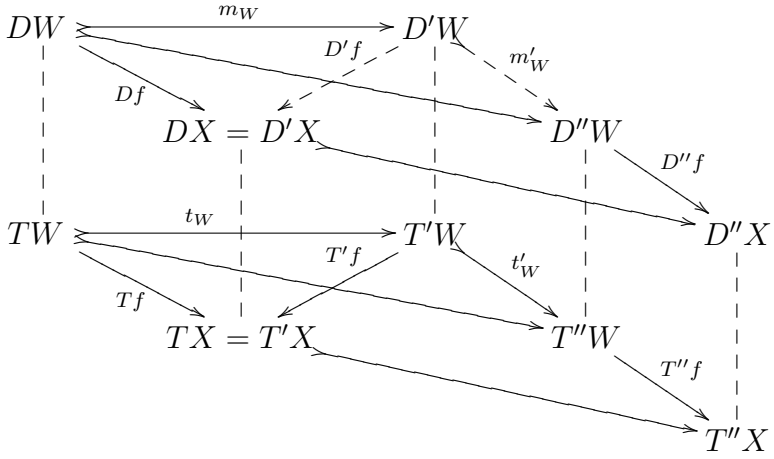
We need some notation for this section: $V : \mathbb{V} \longrightarrow Q\mathbb{E}$ is a view (= sketch morphism); T, T' and T'' are models of $\mathbb{V}$; t' and t'' are insert or delete updates of T at W; D, D' and D'' are models of $\mathbb{E}$ such that $T = V^*D$ and $T' = V^*D'$, $T'' = V^*D''$. When dealing with inserts we will suppose $t : T \rightarrowtail T'$, $m : D \rightarrowtail D'$ and $m'' : D \rightarrowtail D''$ are insert updates. Deletes will be treated dually ($t : T' \rightarrowtail T$ etc). In either case we suppose that $V^*m = t$ and $V^*m'' = t''$.

We assume that V is a sketch morphism whose underlying graph morphism is injective on nodes and edges, and that V^* carries monomorphisms of models to monomorphisms. We begin by considering cases where V is just a view of a part of $\mathbb{E}$. In fact, our first result is closely related to Proposition 19. The explicit construction given below also exemplifies the proof of Lemma 15.

Proposition 21 *Suppose that all of the arrows out of VW in $\mathbb{E}$ are in the image of V, and that VW is* not
- the initial node in any commutative diagram in $\mathbb{E}$
- in any cone or cocone of $\mathbb{E}$
then V is insert updatable at W. Further, if all of the arrows into VW in $\mathbb{E}$ are in the image of V, then V is delete updatable at W.

Proof. We prove only the insert case. The delete case is a straightforward dual argument, except that there is no need to be concerned about commutative diagrams (deleting an element cannot spoil commutativity but inserting an element can).

Let D' be defined by $D'X = DX$ for X not equal to W and $D'W = T'W$, and $D'f = Df$ for arrows f not incident at VW, $D'f = T'f$ for arrows f out of VW, and $D'f = t_W Df$ for arrows f into VW. The natural transformation m has the evident identities and inclusion as components. Consider the following diagram:



Now D' is a model since the limits and colimits are the same as in D and commutativity cannot be spoiled because arrows into $D'W$ factor through DW and naturality of t ensures that for arrows f and g composed through W, $D'fD'g = D'fm_W Dg = DfDg$. Furthermore $m : D \rightarrowtail D'$ is initial among the appropriate $m' : D \rightarrowtail D''$ since it is initial at each component. ∎

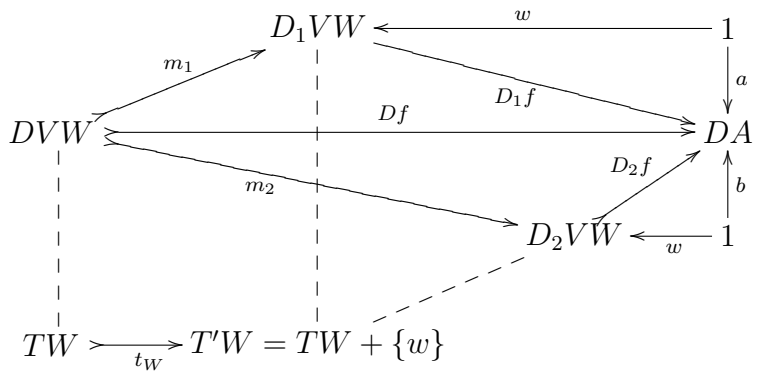
Notice that if VW has a non-trivial attribute A which is not in the image of V, then inserts into W are not updatable since there is no canonical way to assign a value in A to new instances of VW. On the other hand, Proposition 21 says that a view which can "see enough" is updatable. For example, if the view sketch were to include Medical practitioner, Practice agreement, and Hospital, along with the two arrows between them (see Figure 2), then the view is insert updatable, but not delete updatable, at Practice agreement.

For several of the propositions below W is the only entity in the view sketch, and $\mathbb{E}$ is assumed to be very simple. Recall that the "single entity view" is the only meaning of "view" in SQL, for example. In such cases the view sketch can have only a single entity and no other data and we write $\mathbb{V} = \{W\}$ with its image VW under V in $Q\mathbb{E}$. In our applications we encourage larger structured $\mathbb{V}$, but the following propositions are nevertheless still useful then because we can search for parts of $\mathbb{V}$ which match the premises of the propositions and either find a counterexample to updatability at W, or partition $\mathbb{V} - \{W\}$ and argue that updatability for each partition as a view, whether concurrently or serially, implies the updatability of the view V at W. Similarly the propositions can be applied to larger complex

$\mathbb{E}$ because updatability is a "local" phenomenon: Inserts or deletes at W will be updatable according to whether they are updatable in the restriction of $\mathbb{E}$ to objects "near" W.

Proposition 22 *Suppose that* $\mathbb{V} = \{W\}$*, and* $\mathbb{E}$ *has a graph including* $f : VW \longrightarrow A$ *where* A *is a non-trivial attribute, that is, the vertex of a cocone with at least two elements. Then* V *is not insert updatable at* W*.*

Proof. Choose two distinct elements $a, b : 1 \longrightarrow A$. We can make the insert non-trivial and atomic so there is an element $w : 1 \longrightarrow T'W$ which is not in TW and $T'W = TW + \{w\}$ (recall $T1 = 1$). Consider D_1 and D_2 defined by $D_1VW = D_2VW = T'W$, and of course $D_1A = D_2A = DA$ (attributes are constant for all models), with $(D_1f)w = a$ and $(D_2f)w = b$ and $D_1f = D_2f = Tf$ when restricted to TW as in



Each of D_1 and D_2 is clearly a minimal candidate for D' as required by the definition (since we must have $D'VW = T'W$). But there is neither $D_1 \longrightarrow D_2$ nor $D_2 \longrightarrow D_1$ (over D as required). Indeed, suppose there were $\phi : D_1 \longrightarrow D_2$. Then we would have

$$a = D_1f(w) = D_2f\phi_{VW}(w) = D_2f(w) = b$$

where the second and third equalities follow by naturality and constancy of models on attributes. But this is impossible since a and b are distinct. Moreover, any other minimal candidate must be incomparable with one of D_1 or D_2. Thus there is no cocartesian arrow for t_W and the view update is not propagatable. ∎

This proposition apparently implies that for updatability at an entity W, the view must include all of its attributes. To simplify matters we will rather assume for the remainder of this section that W has no attributes. However, as we note in Example 24, the propositions

can be immediately generalized to arbitrary W provided all of the attributes of W do appear in $\mathbb{V}$.

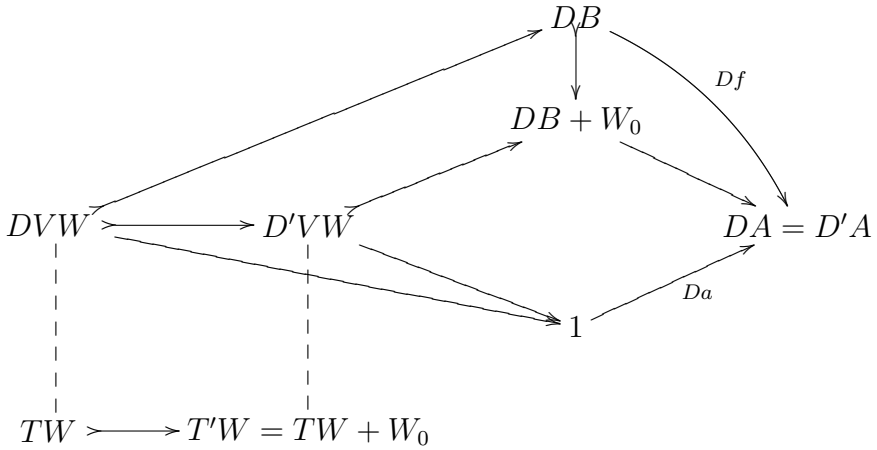
The next proposition is the first in which W is a non-trivial query based on $\mathbb{E}$. These are essentially selection queries.

Proposition 23 *Suppose that $\mathbb{V} = \{W\}$, and $\mathbb{E}$ has as graph $f : B \longrightarrow A$ and an element $a : 1 \longrightarrow A$. Let VW be specified to be the pullback of the arrows f and a as in*

$$\begin{array}{ccc} VW & \longrightarrow & B \\ \downarrow & & \downarrow f \\ 1 & \xrightarrow[a]{} & A \end{array}$$

Then V is insert updatable at W.

Proof. Writing $+$ for disjoint union in $\mathbf{set}_0$, we define W_0 so that $T'W = TW + W_0$ with t the inclusion of the first summand. We can do this since T' is an insert update of T. Let $D'B = DB + W_0$, and $D'A = DA$. Define $D'f$ to be the function whose components on $D'B$ are Df and the constant at a (that is the unique function $W_0 \longrightarrow 1$ composed with the element $a : 1 \longrightarrow A$). Then D' is a model, $m : D \rightarrowtail D'$ is given by the evident inclusion and identity. Calculating the pullback in $\mathbf{set}_0$ shows $D'VW = DVW + W_0$, so $V^*D'W = T'W$ and $V^*m = t$. That is, in the upper part of the following diagram, the inner diamond is also a pullback:



It should be pointed out here that it is not the case that VW is in $\mathbb{E}$, but it clearly does lie in $Q\mathbb{E}$ and this gives meaning to DVW above by the extension of D from $\mathbb{E}$ to $Q\mathbb{E}$.

Now suppose $m'' : D \rightarrowtail D''$ where D'' is another model for $\mathbb{E}$ and $V^*m'' = t't$ for some insert update $t' : T' \longrightarrow T''$. Then there is a unique natural transformation $m' : D' \longrightarrow D''$ commuting with m and m'' since with $D''1 = 1$ and with DA and DB fixed inside $D''A$ and $D''B$, it follows that $D''f$ must have as fiber over a, $(Df)^{-1}(a) + W_0$ in order for the pullback to be $T'W$. These fully determine the components on m'. Thus, V is insert updatable. ∎

Before proceeding, we note that this result (together with the fact the view is also delete updatable) means that the functor

$$V^* : \mathrm{Mon}(\mathbb{E}) \longrightarrow \mathrm{Mon}(\mathbb{V})$$

is a left and right fibration in the case above, and so we have found an instance of universal view updatability. As such, this is an important proposition. At first it might seem surprising that V is insert updatable since the arrow $VW \longrightarrow B$ is rather like that in Proposition 22. But the fact that VW arises as a pullback determines the values that the function must take, and that all those values must be fibered over a. This Proposition is also important because it is an example of an update that many view implementations would not allow despite its practical importance. An example which arises naturally is the following.

Example 24 Take the square from Figure 2, and consider it as an EA sketch. Remember that attributes are not shown in Figure 2. Consider a view consisting of all of the specialists with a given specialization, say all obstetricians. In formal terms, in this view the view sketch $\mathbb{V}$ has one entity Obstetrician, together perhaps with some attributes, and the view schema V is just the inclusion of that entity (and those attributes) into the theory $Q\mathbb{E}$. The image of Obstetrician in $Q\mathbb{E}$ is the limit of

$$1 \longrightarrow \mathsf{College} \longleftarrow \mathsf{Medical\ practitioner}$$

where the first arrow picks out the College of Obstetricians, and the second arrow is member. (This limit is an example of a pullback used to determine a simple join query.)

This view is delete updatable. Further, if all attributes of Medical practitioner appear in the view (as attributes of Obstetrician) then the view is insert updatable.

Notice that the results are independent of the states and attributes of Specialization and College. Many systems will not support inserts for this view, since those systems would require the user to specify the specialization of each newly inserted obstetrician (even though it is always obstetrics) and this cannot be done in a view which does not include Specialization.

This example also provides a good opportunity to show how to deal with key attributes. Suppose that Medical Practitioner has a key attribute MPNbr (for practitioner number). The view just described is not then updatable as it stands, for the view sketch gives no indication

of how to assign an MPNbr to an inserted obstetrician. However, if in the view sketch we replace the attribute MPNbr with a new key attribute ObsNbr for Obstetrician we can recover updatability. To see this we need to specify the relationship between ObsNbr and MPNbr. We require that the key attribute for obstetricians be a summand of MPNbr i.e. it takes part in a discrete cocone. Furthermore, we require that that summand be reserved exclusively for the use of obstetricians (in many actual database examples this is achieved by prefixing the ObsNbr values with say 'O' and other MPNbr values with other letters).

If the hypotheses of Proposition 23 were generalized to replace 1 by an entity C the proposition would no longer hold. However, if C is included in the view sketch along with the pullback VW and the arrow between them then we recover insert updatability.

Alternatively, the hypotheses can be generalized to allow C in place of 1, but strengthened to require that the arrows $C \rightarrowtail A$ and $B \rightarrowtail A$ be monic. In that case V is again insert updatable. The following example addresses some of these issues.

Example 25 Consider the same EA sketch (the square from Figure 2), and suppose the view consists of all specialists from possibly several specializations, perhaps obstetrics, pediatrics and orthopedics. Suppose further that the view sketch includes the subattribute of an attribute of Specialization determined by the image of the chosen specializations on that attribute and suppose that in the current state that attribute takes distinct values for each of the chosen specializations.

The formal definition of the view is little changed: $\mathbb{V}$ still has one entity, and associated attributes including this time the attribute of Specialization. The view schema V is still the evident inclusion. The image of the entity in the theory $Q\mathbb{E}$ is now the limit of

$$[n] \longrightarrow \text{Specialization} \longrightarrow \text{College} \longleftarrow \text{Medical practitioner}$$

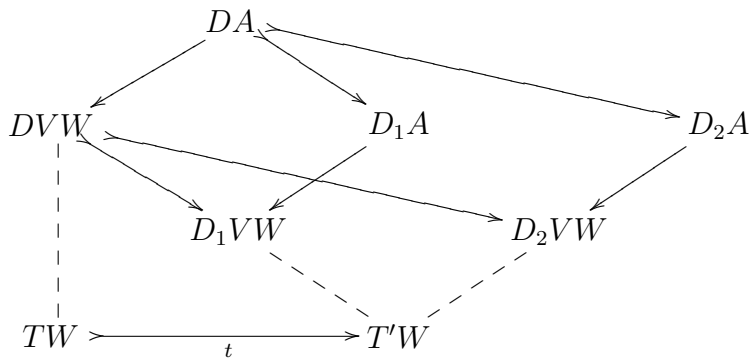
where n is the number of specializations viewed, the first arrow picks out each of the corresponding colleges, and the last arrow is still member.

This view is delete updatable. If all attributes of Medical practitioner appear in the view (as attributes of the viewed entity) then inserts are propagatable *for the current state*. If the viewed attribute of Specialization is guaranteed to take unique values, for example if it is a key, then inserts will be propagatable *for all view states* and so the view will be insert updatable. Conversely, if in some view state the viewed attribute of Specialization did not take unique values on the chosen specializations, then inserts would not be propagatable for that view state.

The next set of results concern updates on disjoint union views.

Proposition 26 *Suppose that $\mathbb{V} = \{W\}$, and $\mathbb{E}$ has only the two entities A and B. Let VW be the sum of A and B. Then V is not insert updatable.*

Proof. The technique we use is similar to that of Proposition 22. We suppose that the insert consists of a single instance $\{w\}$ so that $T'W = TW + \{w\}$. Now we define $D_1A = DA + \{w\}$ and $D_1B = DB$ so that $D_1A + D_1B = D_1VW = T'W$. (Notice that we implicitly extended D_1 from $\mathbb{E}$ to $Q\mathbb{E}$, giving meaning to D_1VW.) We define D_2 similarly, but with $D_2B = DB + \{w\}$, and giving $D_2VW = T'W$. Suppressing the "B-side" and most arrow labels we have the following picture:



But now both D_1 and D_2 are clearly minimal with the property that $D_iVW = T'W$, and so there should, for example, be $D_1 \rightarrowtail D_2$, but this would require $D_1A \rightarrowtail D_2A$ which is impossible (both are finite sets, but D_1A has smaller cardinality). There is no difficulty to extend this argument to inserts of multiple instances into $T'W$, and the result follows. ∎

Proposition 27 *Suppose that $\mathbb{V}$ has underlying graph $A_0 \longrightarrow W$ and no other data, and that $\mathbb{E}$ has only the two entities A and B. Let $VA_0 = A$ and let VW be the sum of A and B. Then V is insert updatable (at W).*

Proof. Unlike the previous proposition, in this case an element of $T'W$ that is not in TW either corresponds to an element of $T'A_0$ or it does not. In the first case we define D' by adding the element to DA, and in the second case by adding it to DB. (As it was described at the beginning of this section, a strict reading of "insert at W" would mean that only the second case could arise.) In either case it is now easy to see that the D' so constructed is initial, and the view is insert updatable at W. ∎

Again this (with the similarly proved delete updatability) means that the functor

$$V^* : \mathrm{Mon}(\mathbb{E}) \longrightarrow \mathrm{Mon}(\mathbb{V})$$

is a left and right fibration.

Example 28 Here we consider the effect of having Medical practitioner specified to be the sum of GP and Specialist.

If only the two triangles are taken as the EA sketch, then Medical practitioner is just an ordinary entity, so a view with Medical practitioner as its only entity is both insert and delete updatable (at the unique entity).

If the two triangles plus GP and Specialist, together with the sum specification, are taken as the EA sketch, then a view sketch with Medical practitioner as its only entity defines a view that is delete updatable, but not insert updatable. If the view sketch includes both Medical practitioner and at least one of GP or Specialist then the view is both insert and delete updatable

Proposition 29 *Suppose that* $\mathbb{V} = \{W\}$*, and* $\mathbb{E}$ *has two entities* A *and* B*. Let* VW *be the product of* A *and* B*. Then* V *is not insert updatable.*

Proof. As noted in Section 4 if, as is usually the case, adding 1 to TW leads to its number of elements being coprime to the number of elements in DA and in DB then there are no models D' such that $V^*D' = T'$ and a fortiori no initial such, and hence the view insert is not propagatable. ∎

For the record, the hypotheses of all but the last proposition result in delete updatability.

We are now in position to explain our position on universal view updatability for the parts of the query language $Q\mathbb{E}$ on an EA sketch which are similar to the main operations of relational algebra. Proposition 23 and the comments following it concern `join` views. With the restrictions noted, join views should be universally insert updatable. The `select` views are usually considered to be updatable and we also find them to be universally insert updatable. Of course, product views are not updatable (Proposition 29), and similar arguments quickly show that `project` views cannot be updatable. We treat the case of `union` via sum views, that is disjoint union, and here the results of Propositions 26 and 27 show that while a disjoint union view is not universally updatable, a disjoint union *plus injection* view is updatable.

Thus, in the sketch data model we have shown universal updatability for a class of views which is larger than that allowed under languages such as SQL. On the other hand we do

not find universal updatability for some of the views which would be considered updatable in recent editions of Date's text (see [16]). For example, he would often allow insertion in a union view (without injections) by prescribing insertion in both of the summands whereas such an insertion is not minimal (insertion into either summand would suffice) and so there is no canonical choice of underlying database update. We emphasize that this difference is not a failure of the theory presented here. Rather, it recognizes that Date's proposal is ad hoc. Indeed, as Date himself notes ([16] page 308) his "solution" sometimes gives "slightly surprising results". He goes on to justify these by claiming that "such surprises can occur only if the database is badly designed". We have chosen instead to develop a theory of updates which by virtue of their universal properties will never be surprising.

7 Related Work

In this section we begin by briefly noting some work that has applied category theory to data modelling and then we turn to more traditional database work which has inherent in it some category theoretic ideas. After that we describe related view update research before considering the connections between that research and the present paper. Finally we note some other developments in this programme of research.

In the last decade there has been considerable use of sketches to support semantic data modelling. Piessens and Steegmans developed a notion of data specification including sketches. They have obtained results on the algorithmic determination of equivalences of model categories [38, 39] which were intended to support plans for view integration. Diskin and Kadish ([18] and other articles) have used sketches for a variety of modelling purposes. They have concentrated on developing a diagrammatic language of "diagram operations". Several others, including Lippe and ter Hofstede [36], Islam and Phoa [24], Tuijn and Gyssens [44], Rosebrugh and Wood [41] and Baclawski et al. [3], have also applied category theory to data modelling.

Work on updating problems includes the paper of Atzeni and Torlone [2] who developed a solution to the problem of updating relational databases through weak instance interfaces. While they explicitly discuss views, and state that their approach does not deal with them, the technique for obtaining a solution is analogous to the technique used here. They consider a range of possible solutions, (as we here consider the range of possible updates) and they construct a partial order on them, and seek a greatest lower bound (analogously with our universal solution). A similar approach, also to a non-view problem, appears in Lecluse and Spyratos [35].

The article of Bancilhon and Spyratos [4] remains influential. They treat a database as an arbitrary set C, meant to specify its states and consider a view to be an arbitrary surjective

mapping $f : C \longrightarrow A$. They consider a "complete set of updates", a set U of endo-mappings of A, which is one that is closed under composition and contains a left inverse for every endo-mapping. The idea is that the view updates must be composable and reversible. They look for a "translator" which is an assignment of a lift t_u to C for each u in U, so $ft_u = uf$. They obtain a "translator under constant complement" the complement being a function $g : C \longrightarrow B$ so that $\langle f, g \rangle$ is monic.

Gottlob, Paolini and Zicari [22] extend the work of Bancilhon and Spyratos to a class of views which they call *consistent*. They distinguish *static* views (mere views) from what they call *dynamic* views for which a view definition comes equipped with a specific choice of update policy.

Recently Lechtenbörger [34] has revisited Bancilhon and Spyratos' "constant complement" approach to view updating and related it to the reversibility of updates.

The approach of Bancilhon and Spyratos has now been elucidated by Bohannon et al [7]. 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 $f : C \longrightarrow\!\!\!\!\!\rightarrow A$ and a (partial) function $A \times C \longrightarrow\!\!\!\!\!\rightarrow C$ satisfying suitable axioms. The idea is that a database state and an *updated* view state determine a database state—the updated database state for the updated view state. More recently Foster et al [20] have developed the lens approach to apply it to tree structured data and also include a valuable review of the view updating literature.

Dayal and Bernstein [17] work in detail in the relational model and introduce *monotonicity* as a criterion for update translation. Monotonicity simply requires that when a view is updated by for example an insertion, the view update should be carried out on the underlying database by an insertion too. Similarly for deletions. Furthermore, and more interestingly in the light of the present article, they advocate considering unique and minimal update translations as the solutions to view update problems, but they argue that this is in fact too restrictive in their context and weaken the uniqueness requirement.

Hegner [23] treats the states of a database as an ordered set D and then defines a view to be a surjective monotone mapping $\gamma : D \longrightarrow V$ such that γ reflects the order (he calls these "open surjections"). For Hegner a "closed update family" is an order-compatible equivalence relation on the ordered set of database states. The idea is that related states may be updated to one another, symmetry expressing reversibility of updates and transitivity that updates may be composed. Hegner defines an "update strategy" to be a function $V \times D \longrightarrow\!\!\!\!\!\rightarrow D$ satisfying certain axioms. It is very much like a lens.

We now turn to how the cited previous work compares with the techniques presented in this paper. First we note that many of the authors have treated the states of a database as an abstract set. This group includes Bancilhon and Spyratos [4], Lechtenbörger [34] and Bohannon et al [7]. In contrast, Dayal and Bernstein [17] and Hegner [23] consider ordered sets of states and operations that respect the order. This is in the same spirit as the work

presented here which carries the idea further to consider the *category* of database states. The fact that Dayal and Bernstein [17] considered using unique and minimal translations for view updating is a very early example of a proposed universal property. Also, their restriction to monotonic updating is carried through in the work presented here—we only search among possible insert updates for the universal insert update. The work of Gottlob, Paoli and Zicari [22], like ours (as shown in [32]), provides more general solutions than the constant complement approach of Bancilhon and Spyratos. They do so by weakening *constant* complements to $\leq$-*shrinking* complements. An exploration of the precise relationships between constant complements, $\leq$-shrinking complements and universal updates is part of ongoing work [32]. One of the particular novelties of the work presented here is the fact that in the category theoretic setting solutions to view update problems can arise from universal properties, and queries themselves arise from universal properties. The interplay between these two leads to a more coherent treatment permitting results like those given in Section 6.

In other work related to this article the authors and collaborators have further tested the techniques presented here. Johnson and Dampney [13] have used the techniques in a case study. Johnson, Rosebrugh and Wood [33] developed a mathematical foundation that unifies the treatment of the categories of database states, the query language and updates as instances of modelling sketches in 2-categories. They first defined EA sketches and studied the categories of models of EA sketches in a lextensive category. In addition they showed that both the updates of the models of an EA sketch and the query language of an EA sketch are models of the EA sketch in other 2-categories. The current authors [26] show how the techniques can be used for database interoperability. They have also developed algorithms implementing updates for certain classes of EA sketches, reported in the articles [27] and [28], and they have studied how to deal with missing information, finding that three possible methodologies are equivalent under suitable restrictions [31].

8 Conclusion

After introducing the sketch data model in Section 2, we defined views in Section 3. We start with a view sketch. A view schema is then a morphism of EA sketches from the view sketch to the (sketch of queries on the) underlying database schema. This offers maximum generality because the view schema can refer to any data that can be obtained from queries on the underlying database. Using these ideas we defined a solution to the view update problem in Section 4 in terms of a universally defined update. We showed in Sections 5 and 6 that we can also obtain results about the updatability of schemata. We emphasize that the viewpoint adopted here led to the universal property for view updatability being based initially on models, rather than the usual schema based definitions.

The work presented here has some limitations:

1. Views take values in $Q\mathbb{E}$ which contains all structural queries, but no arithmetic or collection queries that could summarize, rather than extract and manipulate, data.

2. The updates dealt with are only insert and delete updates. We do not treat modifications of data *in situ*.

3. There is no special treatment of nulls (in agreement with, for example, Date's recommendation [16] that systems should not support nulls).

4. We have not presented detailed consideration of implementation issues. In particular the treatment of both the *when* and the *how* of view updating by universal properties does not directly address implementation issues. But see below.

5. We have not considered here how to translate an EA sketch into a relational schema with constraints. Preliminary work on this and the relation of our sketch data model with that of Piessens and Steegmans is in the article [30].

Each of these is the subject of ongoing research. In particular, implementations are in progress (see for example [40], an EA-sketch compiler which supports graphical manipulation of EA-sketches and automatic conversion into Oracle and MySQL databases). It should be noted however that implementations of universal view updates are not proposed at this time to include algorithms for *finding* universal solutions (and so are not subject to results like Buneman's [9] NP-hardness results). Instead, the technique that we have used in industrial consultancies has been to analyze mathematically each view to determine whether or not there is a universal solution. The result of the analysis is either a counterexample, in which case the consultant negotiates with the client to improve the database design, or a theorem in the style of Section 6. The proof of the universal view update theorem encodes the implementation information that can be used in the particular system to *calculate* updates as required.

The new approach to views has significant advantages:

1. The sketch data model is seen to incorporate views very naturally. This allows data based on any structural query to be viewed in a sketch data model schema, subject only to compatibility with the underlying information system as implied by the view V being a sketch morphism.

As expected, we cannot constrain the data in the view more than it is constrained in the underlying database since the former is derived from the latter.

2. View updatability is defined in a consistent framework based on a universal property. We believe that this gives the most general reasonable definition of view updatability.

3. The closure obtained by having a view be itself a (morphism of) sketch data model(s) intrinsically allows views of views (composite views) etc. It also supports proposals for using views as the interface for database interoperability and for federated information systems (see [26]) .

4. The propositions presented in Sections 5 and 6 allow us to work with schema updatability (rather than model based updatability) in the usual way, and as noted above the proofs of the propositions provide the code required to carry out the update.

Finally, we note that it is presumably the appeal of schema based view updatability that has drawn previous workers into defining updatability in terms of schemata. We would argue that it is this that has led to view updatability being seen as particularly problematic.

Acknowledgements

The authors are pleased to thank the anonymous referees and the editor, Benjamin Pierce, for insightful and helpful comments.

References

[1] Kato, Akihiko. An abstract relational model and natural join functors. *Bull. Inform. Cybernet.* 20, 95–106, 1983.

[2] P. Atzeni and R. Torlone. Updating relational databases through weak instance interfaces. *ACM Trans. Database Syst.* 17, 718–743, 1992.

[3] K. Baclawski, D. Simovici and W. White. A categorical approach to database semantics. *Mathematical Structures in Computer Science* 4, 147–183, 1994.

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

[5] M. Barr and C. Wells. *Category theory for computing science*. Prentice-Hall, second edition, 1995.

[6] M. Barr and C. Wells. *Toposes, Triples and Theories*. Grundlehren Math. Wiss. 278, Springer Verlag, 1985.

[7] A. Bohannon, B. Pierce and J. Vaughan. Relational Lenses: A language for updatable views. *Proceedings of ACM PODS-2006*, 2006.

[8] F. Borceux. *Handbook of Categorical Algebra 3*. Cambridge University Press, 1994.

[9] P. Buneman, S. Khanna and W. Tan. On propagation of deletions and annotations through views. *Proceedings of ACM PODS-2002*, 150–158, 2002.

[10] P. P. -S. Chen. The Entity-Relationship Model—Toward a Unified View of Data. *ACM Trans. Database Syst.* 2, 9–36, 1976.

[11] C. N. G. Dampney and Michael Johnson. TIME Compliant Corporate Data Model Validation. Consultants' report to Telecom Australia, 1991.

[12] C. N. G. Dampney and Michael Johnson. Fibrations and the DoH Data Model. Consultants' report to NSW Department of Health, 1999.

[13] C. N. G. Dampney and Michael Johnson. A formal method for enterprise interoperability: A case study in a major health informatics information system. *Proceedings of the Thirteenth International Conference on Software and Systems Engineering*, CNAM Paris, Vol 3, 12-5, 1–6, 2000.

[14] C. N. G. Dampney, Michael Johnson and G. M. McGrath. Audit and Enhancement of the Caltex Information Strategy Planning (CISP) Project. Consultants' report to Caltex Oil Australia, 1994.

[15] C. N. G. Dampney, Michael Johnson, and Robert Rosebrugh. View Updates in a Semantic Data Model Paradigm. *Proceedings of the Twelfth Australasian Database Conference ADC2001*, 29–36, IEEE Press, 2001.

[16] C. J. Date. *Introduction to Database Systems.* Addison-Wesley, eighth edition, 2004.

[17] U. Dayal and P. A. Bernstein. On the correct translation of update operations on relational views. *ACM Trans. Database Syst.* 7, 381–416, 1982.

[18] Zinovy Diskin and Boris Kadish. Variable set semantics for keyed generalised sketches: formal semantics for object identity and abstract syntax for conceptual modeling. *Data and Knowledge Engineering* 47, 1–59, 2003.

[19] H. Ehrig and B. Mahr. *Fundamentals of algebraic specifications.* Springer-Verlag, 1985.

[20] J Foster, M. Greenwald, J. Moore, B. Pierce and A. Schmitt. Combinators for bi-directional tree transformations: A linguistic approach to the view update problem. *ACM Transactions on Programming Languages and Systems*, to appear, 2006.

[21] M. Gogolla and U. Hohenstein. Towards a semantic view of an extended entity-relationship model. *ACM Trans. Database Syst.* 16, 369–416, 1991.

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

[23] S. J. Hegner. An order-based theory of updates for closed database views. Annals of Mathematics and Artificial Intelligence, 40, 63–125, (2004).

[24] A. Islam and W. Phoa. Categorical models of relational databases I: Fibrational formulation, schema integration. *Proceedings of the TACS94.* Eds. M. Hagiya and J. C. Mitchell. *Lecture Notes in Computer Science* 789, 618–641, 1994.

[25] Michael Johnson and C. N. G. Dampney. On the value of commutative diagrams in information modelling. In Algebraic Methodology and Software Technology, *Springer Workshops in Computing*, 1994.

[26] Michael Johnson and Robert Rosebrugh. Database interoperability through state based logical data independence. *International Journal of Computer Applications in Technology* 16, 97–102, 2003.

[27] Michael Johnson and Robert Rosebrugh. Coproducts in Categorical Information System Specification. *Proceedings of SCI 2001*, Vol XIV, 145–150, July 2001.

[28] Michael Johnson and Robert Rosebrugh. Update algorithms for the sketch data model. *Proceedings of CSCWD 2001, the Sixth International Conference on Computer Supported Cooperative Work in Design*, 367–376, NRC and IEEE Canada, London, Ontario, July 2001.

[29] Michael Johnson and Robert Rosebrugh. View updatability based on the models of a formal specification. *Proceedings of Formal Methods Europe 2001*, 534–549, *Lecture Notes in Computer Science* 2021, 2001.

[30] Michael Johnson and Robert Rosebrugh. Sketch data models, relational schema and data specifications. Proceedings of CATS '02, *Electronic Notes in Theoretical Computer Science* 61(6), 1–13, 2002.

[31] Michael Johnson and Robert Rosebrugh. Three approaches to partiality in the sketch data model. Proceedings of CATS '03, *Electronic Notes in Theoretical Computer Science*, 78, 1–18, 2003.

[32] Michael Johnson and Robert Rosebrugh. Complements and universal view updates. In preparation.

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

[34] J. Lechtenbörger. The impact of the constant complement approach towards view updating. *Proceedings of ACM PODS-2003*, 49–55, 2003.

[35] C. Lecluse and N. Spyratos. Implementing queries and updates on universal scheme interfaces. *Proceedings of VLDB*, 62–75, 1988.

[36] E. Lippe and A. ter Hofstede. A category theoretical approach to conceptual data modelling. *RAIRO Theoretical Informatics and Applications* 30, 31–79, 1996.

[37] B. C. Pierce. *Basic category theory for computer scientists*. MIT Press, 1991.

[38] F. Piessens and Eric Steegmans. Categorical data specifications. *Theory and Applications of Categories* 1, 156–173, 1995.

[39] F. Piessens and Eric Steegmans. Selective Attribute Elimination for Categorical Data Specifications. *Proceedings of the 6th International AMAST* 424–436, *Lecture Notes in Computer Science* 1349, 1997.

[40] R.Rosebrugh, R. Fletcher, V. Ranieri and K. Green. EASIK: An EA-Sketch implementation kit. Available from `http://www.mta.ca/~rrosebru`.

[41] Robert Rosebrugh and R. J. Wood. Relational databases and indexed categories. In *Proceedings of the International Category Theory Meeting 1991, CMS Conference Proceedings 13*, 391–407, American Mathematical Society, 1992.

[42] D. Shipman. The functional data model and the data language DAPLEX. *ACM Trans. Database Syst.* 6, 140–173 1981.

[43] Ross Street. Fibrations in bicategories. *Cahiers de topologie et géometrie différentielle* XXI, 111–160, 1980

[44] C. Tuijn and M. Gyssens. CGOOD, a categorical graph-oriented object data model. *Theoretical Computer Science* 160, 217-239, 1996.

[45] J. D. Ullman and J. Widom. *A first course in database systems*, Prentice-Hall, second edition, 2002.

[46] R. F. C. Walters. *Categories and Computer Science.* Cambridge University Press, 1991.