View Updates in a Semantic Data Modelling Paradigm

Michael Johnson
Department of Computing
Macquarie University
Sydney, Australia
Email: mike@ics.mq.edu.au

Robert Rosebrugh
Department of Math and CS
Mount Allison University
NB, Canada
Email: rrosebru@mta.ca

C. N. G. Dampney
Department of Computing
Macquarie University
Sydney, Australia
Email: kit@ics.mq.edu.au

Abstract

The Sketch Data Model (SkDM) is a new semantic modelling paradigm based on category theory (specifically on categorical universal algebra), which has been used successfully in several consultancies with major Australian companies. This paper describes the sketch data model and investigates the view update problem (VUP) in the sketch data model paradigm. It proposes an approach to the VUP in the SkDM, and presents a range of examples to illustrate the scope of the proposed technique. In common with previously proposed approaches, we define under what circumstances a view update can be propagated to the underlying database. Unlike many previously proposed approaches the definition is succinct and consistent, with no ad hoc exceptions, and the propagatable updates form a broad class. We argue that we avoid ad hoc exceptions by basing the definition of propagatable on the state of the underlying database. The examples demonstrate that under a range of circumstances a view schema can be shown to have propagatable views in all states, and thus state-independence can frequently be recovered.
Keywords: View update, category theory, data model, semantic data modelling

1 Introduction

This is a paper about the *view update problem* in the framework of a new semantic data model, the *sketch data model*.

View updating has long been recognised as important and difficult (see for example [11, Chapter 8]). With the growth of the need for database interoperability and graceful evolution, the importance is even greater. Yet proposed approaches continue to be ad hoc, or incomplete, or require explicit application code support. For a range of recent approaches to views see [5], [13], [19], [20], [21], [29], [1].

Interoperability and evolution have also led to calls for greater semantic data modelling [24] with the power to better model real world constraints. The authors and their coworkers have been developing such a modelling paradigm [16], [18] and Dampney and Johnson have been using it in large scale consultancies [9], [8], [28]. Recently the methodology has come to be called the *Sketch Data Model* (SkDM) as it is based on the category theoretic notion of mixed sketch [3], [4].

In this paper, in the framework of the SkDM, we propose an approach to the view updating problem which is based, unlike previous approaches, on database states (also known as database instances or snapshots). The approach is consistent across the range of different schemata and instances. Despite the approach being instance based, we can prove that for a large number of schemata view updates can be propagated to the underlying database independently of the specific instances involved.

The plan of the paper is as follows. In Section 2 we introduce the sketch data model, illustrating it with an example based on a health informatics model. Section 3 outlines briefly the mathematical foundation of the sketch data model. The foundation is important, although we try to suppress mathematical details as far as possible in the rest of the paper, and we believe that the paper can be understood reasonably well without detailed study of this section. Section 4 sets the view update problem in the framework of the sketch data model by presenting a formal, and very broad, definition of view. In Section 5 we discuss the importance of logical data independence, and use it to motivate our definition of propagatable inserts and deletes in a given view. Section 6 presents a selection of examples of view inserts and deletes and shows that propagatability can frequently be determined for a schema without reference to its instances. Finally, Section 7 reviews related work and Section 8 concludes.

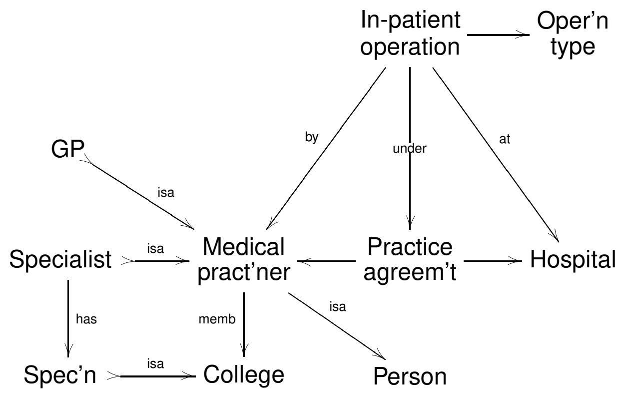

Figure 1. A fragment of a health informatics graph, the main component of a health informatics SkDM

2 The sketch data model

The sketch data model paradigm is a semantic data modelling paradigm, closely related to ER modelling [7] and functional data models [12, 204–207], while incorporating support for constraints via commutative diagrams, finite limits and finite coproducts [3]. Formally a sketch data model is specified by giving an ER sketch. The notion of ER sketch is defined below (Section 3), but in this section we will concentrate on giving an informal presentation of a sketch data model by working through an example (Figure 1).

Figure 1 presents a small fragment of a health informatics graph, chosen for its illustrative value. (Figure 1 is *not* in fact part of the Department of Health data models as they are confidential.) To aid the following discussion we have simplified the model a little.

The affinity with ER modelling should be clear on casual inspection. The graph shows as nodes both entities and relationships, and as arrows certain many-to-one relations (functions). Attributes are often not shown, but may also be included by representing the domain of attribute values as a node and the function representing the assignment of those values as an arrow. Some relationships, such as Practice agreement, are tabulated — represented as two functions — while others, such as isa, can be represented as single functions.

Now to the extra-ER aspects of a sketch data model. Further semantics are incorporated into the model by recording which diagrams commute, and which objects arise as limits or as coproducts of other components of the graph.

A diagram is said to *commute* when the composites of the functions along any two paths with common source and target are required to be equal. Thus, for example, a Specialist isa Medical practitioner who is a member of a College, and a Specialist has a Specialisation which isa College. Naturally we require that the two references to "College" in the last sentence refer, for any single specialist, to the same college — we require the diagram to commute.

Not all diagrams commute, and we have demonstrated repeatedly in consultancies the value of determining early in the design process which diagrams do commute, and why those that do not commute should not do so. The fact that commuting diagrams can be used to model real world constraints (for example business rules) can be seen by considering the two triangles: Their commutativity reflects the requirement that no operations take place without there being a practice agreement between the practitioner and the hospital. If instead the arrow under was not in the model, then Practice agreement would merely record those agreements that had been made. If the arrow was there and the triangles were not required to commute then each operation would take place under an agreement, but it would be possible for example to substitute practitioners — to have one practitioner operate under another practitioner's agreement.

We will not define limits and coproducts here. Instead we refer the reader to a standard text, say [30] or [3]. But we will indicate some uses of limits and coproducts in our example.

Coproducts correspond to disjoint union. They can be used to model logical disjunctions and certain type hierarchies. To take a simple example, requiring that Medical practitioner be the coproduct of Specialist and GP ensures that every practitioner also appears as either, but not both of, a specialist or a GP. The registration details, which are different for specialists and GPs, are recorded as attributes (not shown) of the relevant subtypes.

Limits can take many forms. We will mention just three:

1. The cartesian product is a limit, usually just called the product. It could be used for example to specify a function

$$\mathsf{Specialisation} \times \mathsf{Operation\ type} \longrightarrow \mathsf{Scheduled\ fee}.$$

2. Injective functions can be specified via a limit. The functions in Figure 1 shown as $\rightarrowtail$ are required to be injective, and this is achieved via a limit specification. (A minor, but important point for the model: The arrow into Person is not required to be injective, reflecting the fact that a single person may appear more than once as a medical practitioner, for example, the person might practice both as a GP and as a specialist, or might have more than one specialisation.)

3. A wide range of selection operations can be expressed as pullbacks. For example, specifying that the square

in Figure 1 be a pullback ensures that all and only those practitioners who are members of colleges which occur among the list of specialisations will appear as specialists.

To sum up, a sketch data model $\mathbb{E}$ is a graph, like an ER graph, together with specifications of commutative diagrams, limits and coproducts. We emphasise that this is a very simple structure: all of these notions can be described in terms of a graph with an associative composition of arrows which has identities (such a graph is called a *category*). Yet the specifications have a surprisingly wide range of semantic power. Furthermore, this approach has been demonstrated to be useful in industrial consultancies.

3 Formal definitions for the SkDM

For completeness, this section outlines the mathematical foundation for the sketch data model paradigm. The full details are not essential for understanding the main points of the paper and some readers might wish to skip this section on a first reading.

A *cone* $C = (C_b, C_v)$ in a directed graph $G = (N, E)$ consists of a graph I and a graph morphism $C_b : I \longrightarrow G$ (the *base* of C), a node C_v of G (the *vertex* of C) and, for each node i in I, an edge $e_i : C_v \longrightarrow C_b i$. *Cocones* are dual. The edges e_i in a cone (respectively cocone) are called *projections* (respectively *injections*).

Definition 1 A *sketch* $\mathbb{E} = (G, \mathbf{D}, \mathcal{L}, \mathcal{C})$ is a directed graph G, a set of pairs of paths in G with common source and target $\mathbf{D}$ (called the commutative diagrams) and a set of cones (respectively cocones) in G denoted $\mathcal{L}$ (respectively $\mathcal{C}$).

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

Definition 3 A *model* M of a sketch $\mathbb{E}$ in a category $\mathbf{S}$ is an assignment of nodes and edges of G to objects and arrows of $\mathbf{S}$ so that the images of pairs of paths in $\mathbf{D}$ have equal composites in $\mathbf{S}$ and cones (respectively cocones) in $\mathcal{L}$ (respectively in $\mathcal{C}$) have images which are limit cones (respectively colimit cocones).

To each sketch $\mathbb{E}$ there is a corresponding *theory* [3] or *classifying category* [6] which we denote by $\widetilde{\mathbb{E}}$. Using the evident inclusion $G \longrightarrow \widetilde{\mathbb{E}}$ we can refer to nodes of G as objects, edges of G as arrows and (co)cones of $\mathbb{E}$ as (co)cones in $\widetilde{\mathbb{E}}$.

A model M of $\mathbb{E}$ in $\mathbf{S}$ extends to a functor $\widetilde{M} : \widetilde{\mathbb{E}} \longrightarrow \mathbf{S}$. If M and M' are models a *homomorphism* $\phi : M \longrightarrow M'$ is a natural transformation from $\widetilde{M}$ to $\widetilde{M'}$. Models and homomorphisms determine a category of models of $\mathbb{E}$ in $\mathbf{S}$ denoted by $\mathrm{Mod}(\mathbb{E}, \mathbf{S})$, and it is a full subcategory of the functor category $[\widetilde{\mathbb{E}}, \mathbf{S}]$.

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, A (finite limit, finite coproduct)-sketch is a sketch in which all cones and cocones are finite (the graphs which are the domains of the (co)cone bases are finite graphs), and all the cocones are discrete (the graphs which are the bases of the cocones have no edges, only nodes).

Definition 4 An *ER sketch* $\mathbb{E} = (G, \mathbf{D}, \mathcal{L}, \mathcal{C})$ is a (finite limit, finite coproduct)-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 whose injections are elements are called *attributes*. An attribute is not the domain of an arrow.
- The underlying graph of $\mathbb{E}$ is finite.

In this paper an ER sketch is frequently called *a* sketch data model (while *the* sketch data model refers to the SkDM paradigm).

Definition 5 A *database state* D for an ER 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 $\mathbf{Set}_0$. Thus morphisms of database states are natural transformations.

Remark 6 Notice that every ER model yields an ER sketch: Let G be the ER graph, let $\mathbf{D}$ be empty and let $\mathcal{L}$ contain only the mandated empty cone with vertex 1. Let $\mathcal{C}$ be the set of discrete cocones of elements of each attribute domain. If we want to ensure that the ER-relations are actual mathematical relations, add for each ER-relation a product cone with base the discrete diagram containing the entities that it relates, and a "monic" arrow from the relation node into the vertex of the cone. Add cones to $\mathcal{L}$ to ensure that the "monic" arrows are indeed monic in all models (a pullback diagram for each such arrow will suffice).

It is now easy to see precisely the extra descriptive capabilities of the sketch data model: $\mathbf{D}$ can be used to record constraints, and $\mathcal{L}$ and $\mathcal{C}$ can be used to calculate query results from other objects. These query results can in turn be used to add constraints, etc. Furthermore, the techniques we are using here have been developed with a firm mathematical foundation, much of which was originally developed for categorical universal algebra.

4 The view update problem

Views, sometimes called external models or instances of subschemes, allow a user to query and/or manipulate data which are only a part of, or which are derived from, the underlying database. Our medical informatics graph (Figure 1) represents a view of a large health administration database. It in turn might provide views to an epidemiologist who only needs to deal with the two triangles, with Operation type, and with their associated attributes; or to an administrator of the College of Orthopaedic Surgeons who needs to deal with all data in the inverse image of that college, and not with any of the data associated only with other colleges, hospitals, etc.

The view update problem (VUP) is to determine under what circumstances updates specified in a view can be propagated to the entire database, and how that propagation should take place. The essence of the problem is that *not all views are updatable*, that is, an insert or a delete which seems perfectly reasonable in the view, may be ill-defined or proscribed when applied to the entire database. For example, a college administrator can insert the medical practitioner details for a new 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 (in the inverse image view) details about hospitals, and every practice agreement must specify a hospital.

In order to limit the effect of the view update problem, views have sometimes been defined in very limited ways. For example, allowable views might be restricted to be just certain row and column subsets of a relational database. But generally we seek to support views which can be derived in any way from the underlying database so views might be the result of any query provided by the database, and ought to be able to be structured in any way acceptable under the data model in use.

For the sketch data model we now provide a definition of view which supports the generality just described, and in Section 5 we provide a solution to the view update problem in the sketch data model paradigm, while in Section 6 we give a range of examples to give some indication of the breadth of that solution.

Recall from Section 3 that for each sketch $\mathbb{E}$ there is a corresponding category denoted $\widetilde{\mathbb{E}}$. We observed in [10] that the objects of the classifying category correspond to the structural queries of the corresponding database (structural queries do not include numerical computations like `count()` or `avg()`). This motivates the following definition:

Definition 7 A *view* of a sketch data model $\mathbb{E}$ is a sketch data model $\mathbf{V}$ together with a sketch morphism $V : \mathbf{V} \longrightarrow \widetilde{\mathbb{E}}$.

Thus a view is itself a sketch data model $\mathbf{V}$, but its entities are interpreted via V as query results in the original data model $\mathbb{E}$. In more formal terms, a database state D for $\mathbb{E}$ is a finite set valued functor $D : \widetilde{\mathbb{E}} \longrightarrow \mathbf{Set}_0$, and composing this with V gives a database state D' for $\mathbf{V}$, the V-view of D.

Notation 8 The operation *composing with* V is usually written as V^*. Thus $D' = V^*D$. In fact, V^* is a functor, so for any morphism of database states $\alpha : D \longrightarrow C$ we obtain a morphism $V^*\alpha : D' \longrightarrow V^*C$.

Following usual practice we will often refer to a database state of the form V^*D as a view. Context will determine whether "view" refers to such a state, or to the sketch morphism V. If there is any ambiguity, V should be referred to as the *view schema*.

5 Logical data independence

The definition of view provided in the previous section has wide applicability: The presentation of a view as a sketch data model means it can take any SkDM structural form; the sketch morphism V ensures that the semantics associated to the view by the diagrams, limits and colimits in its sketch data model is compatible with the structure of the underlying database; and the fact that V takes values in $\widetilde{\mathbb{E}}$ allows the view to be derived from any data obtainable from $\mathbb{E}$.

Views thus support *logical data independence* — the logical structure, the design, of a database can change over time, but applications programs which access the database through views will be able to operate unchanged provided only that the data they need is available in the database, and that the view mechanism V is maintained as the underlying database design $\mathbb{E}$ is changed.

We have argued in [17] that view based logical data independence is required for database interoperability, and that it should be provided, as suggested by Myopoulos [24], in a semantically rich model like the sketch data model.

Views, being ordinary database states, albeit obtained as V^*D from some database D, can be queried in the same way as any database. The important question to ask, the view update problem, is "When are views updatable?". After all, logical data independence only works fully when updates to the view can be propagated via the view mechanism to the underlying database.

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

1. There may be no states of the database which would yield the updated view. This usually occurs because the update, when carried to the underlying database, would result in proscribed states. For example, a view schema might include the product of two entities, but only one of the factors. In the view, inserting or deleting from the product seems straightforward, after all, it looks like an ordinary entity with a function to another entity. But in the underlying database the resulting state of the product might be impossible, as for instance if the numbers of elements in the product and the factor become coprime.

2. There may be many states of the database which would yield the updated view. 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, and there are usually many choices.

Since a view is just a database state, we know how to insert or delete instances. Thus we define

Definition 9 We say that a specified view insert/delete is *propagatable* if there is a unique minimal insert/delete on the entire database whose restriction to the view (via V^*) is the given view insert/delete. When an insert/delete is propagatable, we call the database obtained from the unique minimal insert/delete the *propagated update*.

Remark 10 i) In mathematical terms, the definition is: Let $V : \mathbf{V} \longrightarrow \widetilde{I\!E}$ be a view of $I\!E$. Suppose $q : Q \rightarrowtail Q'$ consists of two database states for $\mathbf{V}$ and a database state monomorphism, with Q' being an insert update of Q and with $Q = V^*D$ for some database state D of $I\!E$. We say that the insert q is *propagatable* when there exists an initial $m : D \rightarrowtail D'$ among all those database states $m' : D \rightarrowtail D''$ for which $V^*D'' = Q'$ and $V^*m' = q$. Initial here means an initial object in the full subcategory of the slice category under D. The state D' is then called the *propagated update* (sometimes just the update). The definition of propagatable delete is dual (so we seek a terminal D' among all those $D'' \rightarrowtail D$).
ii) The use of "unique minimal" in the definition does not in general mean a unique state obtained by inserting or deleting a minimal number of elements. An insert update D' is unique minimal among a class of insert updates if for each other update D'' in the class, there is a unique morphism of databases states $\alpha : D' \longrightarrow D''$, respecting the inclusions of the database state D in the updates.
iii) When, as will usually be the case, the database is keyed (that is, for each entity there is a specified attribute called its *key attribute* and a specified injective function from the entity to the attribute) these two interpretations of "unique minimal" do in fact coincide.
iv) Notice that we define when an insert/delete of a view (database state) is propagatable, rather than trying to determine for which view schemata inserts and deletes can always be propagated. Thus, propagatability, view updatability, is in principle dependent on the database state being updated.

In fact we can frequently characterise the updatable states for a given view schema, or even prove that for a variety of view schemata, all database states are updatable. Such results are important for designers so that they can design views that will always be updatable. The next section provides a collection of examples in which this happens.

Definition 11 A view is called *insert* (respectively *delete*) *updatable* when all inserts (respectively deletes) are propagatable, independently of the database state.

6 Examples

We collect here a range of illustrative examples. Generally we keep them (unrealistically) small and simple to better emphasise the point made by each example. The examples are intended to show that the definition given in the previous section does embody an intuitively reasonable notion of propagatability, and to give some indication of the breadth of the applicability of the definition.

Example 12 Take the square from Figure 1, and consider it as a sketch data model. Remember that attributes are not shown in Figure 1. Consider a view consisting of all of the specialists with a given specialisation, say all obstetricians. In formal terms, in this view $\mathbf{V}$ has one entity Obstetrician, together perhaps with some attributes, and the sketch morphism V is just the inclusion of that entity and those attributes into the classifying category generated by the square. The image of Obstetrician in the classifying category 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.)

This view is delete updatable. 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 Specialisation and College, and note that some systems will not support inserts for this view, since those systems would require the user to specify the specialisation of each newly inserted obstetrician (even though it is always obstetrics) and this can't be done in a view which doesn't include Specialisation. Date [12, p153] has argued from this to the need for systems to allow the specification of defaults in view defining fields.

Example 13 Consider the same sketch data model (the square from Figure 1), and suppose the view consists of all specialists from possibly several specialisations, perhaps obstetrics, paediatrics and orthopaedics. Suppose further that the view includes an attribute of Specialisation that in the current database state has unique values for each of the chosen specialisations.

The formal definition of the view is little changed: $\mathbf{V}$ still has one entity, and associated attributes including this time an attribute of Specialisation. The morphism V is still an evident inclusion. The image of the entity in the classifying category is now the limit of

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

where n is the number of specialisations viewed, the first arrow "picks out" each of the corresponding colleges, and the second arrow is still member.

This view is also 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 Specialisation is guaranteed to take unique values, for example if it is a key, then inserts will be propagatable *for all states* and so the view will be insert updatable. Conversely, if in some state the viewed attribute of Specialisation did not take unique values on the chosen specialisations, then inserts would not be propagatable for that state.

Example 14 Take all of the entities in Figure 1 from which College can be reached by following a chain of arrows in the forward direction, that is all entities except Operation type, Hospital and Person, and consider them as a sketch data model in which both the triangle and the square commute. The view data model will be given by taking as $\mathbf{V}$ a diagram of the same shape except that the node corresponding to College, and its two arrows will be missing. Let V send each entity to the inverse image of (say) the Orthopaedics College along the path of arrows connecting the corresponding entity to College. This is the college administrator's view for the Orthopaedics College. The administrator can see all the details, except the Personal details, of all of the members of that college, and no other practitioner's details.

The view is delete updatable (but be careful: if the square is specified to be a pullback then deleting the one instance of Specialisation will in fact delete everything from the view, and correspondingly all members of the college and all of their data from the full database). It is insert updatable at all entities except at In-patient operation and Practice agreement (where the full database needs to know about the associated operation types and hospitals) and at Specialisation (which can have at most one element by the construction of the view). If in the sketch data model for the entire database the square is specified to be a pullback, and Medical practitioner is specified to be a coproduct, then inserts at GP are not propagatable (an insert at GP necessitates an insert at Medical practitioner which necessitates an insert at Specialist after which the coproduct constraint can never be recovered).

Example 15 Let's extend Figure 1 by adding a new subtype of In-patient operation called Under investigation. It will contain those operations which are under investigation as a result of complaints, whether from patients, their families, or practitioners. This extended graph will be our new data model. Let $\mathbf{V}$ consist of two entities and an injective function between them, $A \rightarrowtail B$. Let V take B to the in-patient operations conducted by surgeons who are practicing members of the Orthopaedics College (these instances were in the view in the previous example). Let V take A to the pullback of that inverse image VB along the inclusion of Under investigation into In-patient operation (thus VA is the intersection of VB and Under investigation as subtypes of In-patient operation). This is the view for the investigating board of the Orthopaedics College.

The view is insert and delete updatable. For example an insert into A would, as part of the view, specify which orthopaedic operation was being investigated, and when propagated would generate a new instance in Under investigation corresponding to the operation. This is what would happen when the complaint was first received at the college. (If instead the complaint arrived at say the hospital, a new instance would be inserted into Under investigation and the college investigating board would be able to see it appear in their view.)

In fact, if the view conisted only of A, it would still be insert and delete updatable (provided that it included all the attributes of Under investigation and In-patient operation) although it wouldn't match the semantics of our example very well — inserts into A would propagate to new instances in Under investigation *and* In-patient operation. This would correspond to a complaint arriving about an orthopaedic operation which was not stored in the database. This only seems surprising because we know that the operation must have taken place before the complaint.

If instead we change the semantics to consider, for example, among employees the intersection of those who are senior executives, and those who are medical staff, an administrator who is responsible for hiring senior executive medical staff might employ someone and perform an insert into the intersection (their view) which would propagate successfully to the entities Senior executives, Medical staff and All employees.

Example 16 It is interesting to see what the definition says about data models which include no attributes, so as to see an extreme case of its applicability. Suppose the data model includes no cocones and no cones except the mandated "1" cone. It happens that a view of any single entity in such a data model is insert updatable. The update can be calculated as a Kan extension ([23]) and amounts to freely adding instances related to the inserted instance (rather than seeking extant instances to satisfy obligatory relations). This doesn't work in the presence of attributes because attribute domains are fixed, so we can't freely add new attribute values.

Example 17 Finally, let's consider the effect of having Medical practitioner specified to be the coproduct of GP and Specialist.

If the two triangles are taken as the sketch data model, Medical practitioner is just an ordinary entity, and a view including only Medical practitioner is insert and delete updatable.

If the two triangles plus GP and Specialist, together with the coproduct specification, are taken as a sketch data model, then a view including only Medical practitioner is delete updatable, but not insert updatable. If the view includes both Medical practitioner and GP (or instead Specialist) then it is both insert and delete updatable.

7 Related Work

A number of authors are now using sketches to support data modelling initiatives. Notably Piessens [25], [26] has developed a notion of data specification including sketches. He has since obtained results on the algorithmic determination of equivalences of model categories [27] which are intended to support plans for view integration. Diskin and Cadish have used sketches for a variety of modelling circumstances. See for example [14] and [15]. They have been concentrating on developing the diagrammatic language of "diagram operations".

Atzeni and Torlone [2] have developed a solution to the problem of updating relational databases through weak instance interfaces. Although they explicitly discuss views, and note that their approach does not deal with them, the technique for obtaining a solution is similar to the technique used here. They consider a range of possible solutions (as we here consider the range of possible updates $D \rightarrowtail D''$) and they construct a partial order on them, and seek a greatest lower bound (analogous with our initial/terminal solution). A similar approach, also to a non-view problem, appears in [22].

8 Conclusion

To date our approach, in common with many others, deals with inserts and deletes, but not with modifications of extant values. Also, our views do not contain arithmetic operations, and we have not developed special treatments of null values. Each of these is the subject of ongoing work. Similarly, this paper does not deal with implementational issues which are the subject of ongoing research in computational category theory. Despite these caveats the approach presented here has surprisingly wide applicability.

It seems that a significant part of the difficulty of solving the view update problem has arisen because previous authors have sought a single coherent solution and have based their proposed solutions on schemata. In practice, many particular situations (states) have "solutions" although they fall outside the proposed solution so ad hoc adjustments are made, losing coherence. This has also contributed to the impression that the class of updatable views is difficult to characterise.

We have proposed a single coherent solution, but based it on states, avoiding ad hoc amendments. Although it is based on states, we can prove that many schemata always are or aren't updatable, and we have provided a range of examples of such situations. This gives the benefits that were sought in schema based solutions while avoiding ad hoc amendments.

9 Acknowledgements

The research reported here has been supported in part by the Australian Research Council, the Canadian NSERC, the NSW Department of Health, and the Oxford Computing Laboratory.

References

[1] Serge Abiteboul and Oliver M. Duschka. Complexity of Answering Queries Using Materialized Views. *ACM PODS-98*, 254–263, 1998.

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

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

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

[5] F. Bancilhon and N. Spyratos. Update semantics of relational views. *ACM TODS*, 6:557–575 1981.

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

[7] P. P. -S. Chen. The Entity-Relationship Model—Toward a Unified View of Data. *ACM Transactions on Database Systems*, 2:9–36, 1976.

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

[9] 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, 1993.

[10] C. N. G. Dampney, Michael Johnson, and G. P. Monro. An illustrated mathematical foundation for ERA. In *The unified computation laboratory*, pages 77–84, Oxford University Press, 1992.

[11] C. J. Date. *Introduction to Database Systems*. Addison-Wesley, fourth edition, 1986.

[12] C. J. Date. *Introduction to Database Systems, Volume 2*. Addison-Wesley, 1983.

[13] U. Dayal and P. A. Bernstein. On the correct translation of update operations on relational views. *ACM TODS*, 7:381–416, 1982.

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

[15] Zinovy Diskin and Boris Cadish. Variable set semantics for generalised sketches: Why ER is more object oriented than OO. In *Data and Knowledge Engineering*, to appear, 2000.

[16] Michael Johnson and C. N. G. Dampney. On the value of commutative diagrams in information modelling. In *Springer Workshops in Computing*, Springer-Verlag, 1994.

[17] Michael Johnson and Robert Rosebrugh. Database interoperability through state based logical data independence. To appear in CSCWD2000, the Fourth International Conference on Computer Supported Collaborative Work and Design, IEEE Hong Kong, 2000.

[18] Michael Johnson, Robert Rosebrugh, and R. J. Wood. Entity-relationship models and sketches. Submitted to *Theory and Applications of Categories*, 2000.

[19] A. M. Keller. Algorithms for translating view updates into database updates for views involving selections, projections, and joins. *ACM PODS-85*, 154–163, 1985.

[20] Rom Lagerak. View updates in relational databases with an independent scheme. *ACM TODS*, 15:40–66, 1990.

[21] A. Y. Levy, A. O. Mendelzon, D. Srivastava, Y. Sagiv. Answering queries using views. *ACM PODS-95*, 1995.

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

[23] Saunders Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics 5, Springer Verlag, 1971.

[24] John Myopoulos. Next generation database systems won't work without semantics! Panel session, *SIGMOD Record*, 27:497, 1998.

[25] F. Piessens. *Semantic data specifications: an analysis based on a categorical formulation*. PhD thesis, Katholieke Universiteit Leuven, 1996.

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

[27] F. Piessens and E. Steegmans. Selective Attribute Elimination for Categorical Data Specifications. Proceedings of the 6th International AMAST. Ed. Michael Johnson. *Lecture Notes Computer Science*, 1349:424-436, 1997.

[28] G. Southon, C. Sauer, and C. N. G. Dampney. Lessons from a failed information systems initiative: issues for complex organisations *International Journal of Medical Informatics*, Elsevier Science, 1999.

[29] J. D. Ullman. Information integration using logical views. *ICDT-97*, 1997.

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