View Updatability Based on the Models of a Formal Specification

Michael Johnson[1] and Robert Rosebrugh[2]

[1] Macquarie University, Sydney, Australia,
mike@ics.mq.edu.au
[2] Mount Allison University, NB, Canada
rrosebru@mta.ca

Abstract. Information system software productivity can be increased by improving the maintainability and modifiability of the software produced. This latter in turn can be achieved 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. But, supporting views depends upon a solution to the *view update problem*, and proposed solutions to date have only had limited, rather than comprehensive, applicability. This paper presents a new treatment of view updates for formally specified information systems. The formal specification technique we use is based on *category theory* and has been the basis of a number of successful major information system consultancies. We define view updates by a universal property in a subcategory of models of the formal specification, and explain why this indeed gives a comprehensive 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. The paper is predominantly theoretical, as it develops the theoretical basis of a formal methods technique, but the methods described here are currently being used in a large consultancy for a government Department of Health. Because the application area, information systems, is rarely treated by formal methods, we include some detail about the formal methods used. In fact they are extensions of the usual category theoretic specification techniques, and the solution to the view update problem can be seen as requiring the existence of an initial model for a specification.
Keywords: View update, database, formal specification, information system, category theory, conceptual modelling, data model.

* Research partially supported by the Australian Research Council, NSERC Canada, and the Oxford Computing Laboratory. The authors are grateful for that support, and also for the advice of the anonymous referees.

1 Introduction

Much of the progress of software engineering has been based on limiting the ramifications of software modifications. After correctness, and avoiding gross inefficiencies, producing modifiable code is of prime concern.

In the information systems field, the need for maintainable and easily modifiable software is becoming consistently more important with the dependency on legacy code and the growth of the need for systems interoperability, whether for business-to-business transactions, internet based interfaces, or interdivisional cooperation within an organisation.

Many information systems attempt to address this issue by providing a *view* mechanism. Views allow data derived from the underlying information system to be structured in an appropriate way. Provided that views are used as the interface between an information system and other software, the information system can be modified and, as long as the view mechanism is correspondingly modified, the external software will continue to work.

Typically, the "appropriate" structure for a view is a database interface, allowing data to be queried, inserted, or deleted, as if the system were a standalone database. But importantly, not all view inserts and deletes can be permitted since the view data are derived from the underlying information system, and apparently reasonable changes to the view may be prohibited or ambiguous when applied to the information system. We will see examples below, but for a text book treatment the reader is referred to Chapter 8 of [13]. The *view update problem* is to determine when and how updates of views can be propagated to the underlying information system.

Views have been the subject of considerable research, including for example [22], [23], [29], and [1]. The difficulty of obtaining a comprehensive solution to the view update problem has led to systems which offer only limited view mechanisms. Furthermore, the relatively limited use of formal methods (as opposed to semi-formal methodologies) in information system specification has resulted in views being defined either informally, or solely in terms of the underlying information system's schema. Both of these hamper the use of the view mechanism in facilitating program reuse when the underlying information system changes significantly.

The authors and their coworkers have, over a number of years, been developing a formal method for information system specification based on category theory. The impetus for its development came from a very large consultancy [7] which compelled us to use formal methods to manage the complexity. The techniques we use, recently called the *sketch data model* because they are based upon the category theoretic notion of mixed sketch [3], have since been developed considerably and tested in other consultancies including [10] and [8].

This paper develops a detailed approach to views and view updatability based on the sketch data model. After defining the sketch data model in Section 2 we define views (Section 3) in a manner that is designed to permit a wide range of data accessible from the underlying information system to be structured in the view in any manner describable in the sketch data model.

A solution to the view update problem needs to determine when and how view updates can be propagated to the underlying information system. Typically the *when* is determined by definition — certain view updates are defined to be updatable, and then for those updates the *how* is given by specifying the translation of each view update into an update of the underlying information system. Unfortunately this approach can lead 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. Instead, in Section 4 we define the *when* and *how* together using a universal property [26], and we indicate how the nature of the universal property ensures a certain completeness.

Interestingly the universal property we use is based upon the *models* of the specification when previous solutions have usually defined updatability in terms of schemata (the signatures of the specifications). This is very important for the theoretical development. Nevertheless, in Section 5 we prove a number or propositions that show that, for a range of schemata, view updatability can be determined independently of the models. Such results considerably simplify the application of the theory in industry.

After reviewing related work in Section 6, we conclude by enumerating the limitations and advantages of our approach. We note that it is presumably the appeal of schemata 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 a difficult problem.

2 Category theoretic information system specification

This section provides the mathematical foundation for the sketch data model. It is based on categorical universal algebra, which is the basis of widely used formal method specification techniques [16]. We assume some familiarity with elementary category theory, as might be obtained in [3], [26] or [30]. The graphs we use will always be what have sometimes been called "directed multi-graphs, possibly with loops", see for example [3] page 7. The limits and colimits we will deal with will all be finite in our applications.

Definition 1 A *cone* $C = (C_b, C_v)$ in a graph G 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 (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 $e_i : C_b i \longrightarrow C_v$). The edges e_i in a cone (respectively cocone) are called *projections* (respectively *injections*).

Definition 2 A *sketch* $\mathbb{E} = (G, \mathbf{D}, \mathcal{L}, \mathcal{C})$ consists of a 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 has an underlying sketch: Let G be the underlying graph of the category, $\mathbf{D}$ the set of all commutative diagrams, and $\mathcal{L}$ (respectively $\mathcal{C}$) the set of all limit cones (respectively colimit cocones). Of course underlying sketches are usually not small. The advantage of the theory of sketches is that we can frequently use a sketch to give a finite presentation of an infinite category.

Definition 3 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, 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 4 A *model* M of a sketch $\mathbb{E}$ in a category $\mathbf{S}$ is a graph morphism from G to the underlying graph of the category $\mathbf{S}$ such 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) in $\mathbf{S}$.

Equivalently, a model is a sketch morphism from $\mathbb{E}$ to the underlying sketch of the category $\mathbf{S}$. We can also express models in terms of functors as follows.

To each sketch $\mathbb{E}$ there is a corresponding *theory* [3] which we denote by $Q\mathbb{E}$. The theory corresponding to $\mathbb{E}$ should be thought of as the category presented by the sketch $\mathbb{E}$. For our applications this will be the free category with finite limits and finite colimits generated by the graph G subject to the relations given by $\mathbf{D}$, $\mathcal{L}$ and $\mathcal{C}$, or some subcategory thereof.

Using the evident inclusion $G \longrightarrow Q\mathbb{E}$ we will sometimes refer to nodes of G as objects, edges of G as arrows and (co)cones of $\mathbb{E}$ as (co)cones in $Q\mathbb{E}$. If $\mathbf{S}$ has finite limits and finite colimits then a model M of $\mathbb{E}$ in $\mathbf{S}$ extends uniquely to a functor $QM : Q\mathbb{E} \longrightarrow \mathbf{S}$ which preserves finite limits and finite colimits.

Definition 5 If M and M' are models a *homomorphism* $\phi : M \longrightarrow M'$ is a natural transformation from QM to QM'. Models and homomorphisms determine a category of models of $\mathbb{E}$ in $\mathbf{S}$ denoted by $\mathrm{Mod}(\mathbb{E}, \mathbf{S})$, a full subcategory of the functor category $[Q\mathbb{E}, \mathbf{S}]$.

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. When the specified classes do not include all finite (co)limits we can restrict the theory corresponding to such a sketch to be closed only under the specified limits and colimits and the functor QM will only need to preserve the specified limits and colimits. For example, (finite-product, $\emptyset$) sketches correspond to (multi-sorted) algebraic theories, their theories are categories with finite products and their model functors QM preserve finite products.

Definition 6 An *SkDM 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*. Nodes which are neither attributes, nor 1, are called *entities*.
– The graph of G is finite.

An SkDM sketch is used for specifying information systems. An SkDM sketch is sometimes called *a* sketch data model, while *the* sketch data model usually refers to the sketch data modelling formal methodology.

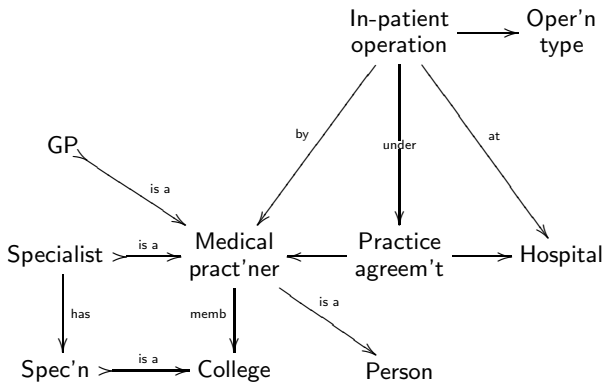

Fig. 1. A fragment of a graph for an SkDM sketch

Example 7 Figure 1 is part of the graph of an artificial SkDM sketch, derived from a fragment of a real sketch data model [8]. The other components of this SkDM sketch are as follows: $\mathbf{D}$ contains both triangles; $\mathcal{L}$ contains the empty cone with vertex 1 (not shown), the square (whose diagonal, one of the projections, is also not shown) and three further cones intended to ensure that the three arrows indicated $\rightarrowtail$ are realised as monics; and $\mathcal{C}$ contains the cocone with vertex Medical practitioner and base Specialist and GP (short for General practitioner), along with a number of cocones with attributes (not shown) as vertices.

Briefly, we expand on each of these components in turn, indicating what they, as a specification, correspond to in models.

– The graph is a type diagram. The three monic arrows indicate subtypes. The other arrows are functions (methods) which given an instance of their domain type will return an instance of their codomain type.
– 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. If, instead the left hand triangle were not required to commute (that is, was not in

D) 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 marked under, and thus they store the contractual information, but do not specify the constraint — it is expected to be added at implementation time (this is one example of why information modelling is *not* usually a formal specification technique).

- The inclusion of the square in $\mathcal{L}$ ensures that in models it will be a pullback. This ensures that the specialists are precisely those medical practitioners who are members of a college which occurs in the subtype Specialisation. This is important because the registration procedures (not shown) for specialists are different from those for other medical practitioners. Similar pullbacks can be used to specify other subtypes, for example, the medical practitioners with a specific specialisation, say otorhinolaryngologists.
- Subtype inclusion arrows, and other arrows that are required to be monic in models, are so specified using pullbacks. Specifically, requiring that

$$\begin{tikzcd} \cdot \arrow[r, "g"] \arrow[d, "g"'] & \cdot \arrow[d, "m"] \\ \cdot \arrow[r, "m"'] & \cdot \end{tikzcd}$$

  be a pullback is another way of saying that m must be monic. Incidentally, we could include just two such cones since elementary properties of pullbacks ensure that if the square is realised as a pullback and its bottom arrow as a monic then its top arrow will necessarily be monic. Notice also that the arrow into Person is not required to be monic. A single person might appear more than once as a medical practitioner, as for example, when the person practises both as a GP and as a specialist, or practises in more than one specialisation. This is a minor point, but the distinction between people and the roles they play is an important distinction in many real world applications.
- The cocone with vertex Medical practitioner ensures that the collection of medical practitioners is the disjoint union of the collection of specialists and the collection of GPs. Specifications which don't include this constraint could be used to allow other vocations, say physiotherapists, to be treated as medical practitioners.
- As is common practice, attributes are not shown in Figure 1, but they are important. They are usually large fixed value sets, often of type `integer` (with specified bounding values), `string` (of specified maximum length), `date` etc. Some examples for this model include the validity period of a practice agreement, the name and the address of a person, the classification of a hospital, the date of an operation, the provider number of a medical practitioner and many more. Strictly, they are all part of the graph, but in practice they are usually listed separately in a data dictionary.

Definition 8 A *database state* D for an SkDM 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 $\mathbf{Set}_0$.

A database state or model of the SkDM sketch of Example 7 is a collection of finite sets and functions satisfying the constraints. The set corresponding to, for example, In-patient operation should be thought of as the collection of all operations currently stored in the information system.

Sketch data modelling can be viewed as an extension of entity-relationship (ER) modelling [6],[28]. An ER diagram can be used to generate in a systematic way a graph for a sketch data model (the details are dealt with in [19]). The theory corresponding to a sketch data model includes objects representing the queries (first noted in [11]). The extra "semantic" power of the sketch data model comes from the non-graph components: $\mathbf{D}$ can be used to specify constraints, and $\mathcal{L}$ and $\mathcal{C}$ can be used to specify the calculation of query results based on other objects, and then these query results can be used to specify further constraints, and so on. It is not surprising that this extra power, by providing a much richer set of possibilities for specifying constraints, is of benefit in information modelling, and it has been the basis of our successful consultancies which have used the sketch data model as a formal information system specification technique.

3 Views

Views are important tools in software engineering. In information systems a view allows a user to manipulate data which are part of, or are derived from, an underlying database. For example 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 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.

Views have generally been implemented in very limited ways so as to avoid difficulties related to the view update problem. For example, allowable views might be restricted to be just certain row and column subsets of a relational database. However, we seek to support views which can be derived in any way from the underlying database, so views might include the result of any query provided by the database, and we argue that views 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.

Recall from Section 2 that for each sketch $\mathbb{E}$ there is a corresponding theory, often called the classifying category, denoted $Q\mathbb{E}$. We observed in [11] that the objects of the classifying category correspond to the queries of the corresponding information system. This motivates the following definition.

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

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 $D : Q\mathbb{E} \longrightarrow \mathbf{Set}_0$, and composing this with V gives a database state D' for $\mathbf{V}$, the V-view of D.

Remark 10 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*.

4 Updatability

We have defined *view* above so as to ensure that views have the widest possible applicability. A view is a sketch data model, and so it can appear in any structural form acceptable to the sketch data model formal specification technique. The use of a sketch morphism V guarantees that the constraints on the view imposed by the diagrams, limits and colimits in the view's sketch data model are compatible with the constraints on the underlying database. And the fact that V takes values in $Q\mathbb{E}$ permits any data derivable from the model of the underlying database to appear in the view.

Views support software maintenance — as long as the view mechanism is maintained, the logical structure (the design) of a database can be changed without needing to modify applications programs which access it through views. The only risk is that needed data might actually be removed from the database. If, on the other hand, the data is there in any form it can be extracted as an object of $Q(\mathbb{E})$ and accessed via a view. The breadth of the definition of view is important to ensure that this support for maintenance can be carried out in the widest possible range of circumstances.

The view update problem is to determine under what circumstances updates specified in a view can be propagated to the underlying information system, 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 underlying information system. For example, a college administrator can 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 (in the inverse image view) 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 [12]:

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.

Thus we define

Definition 11 Let $V : \mathbf{V} \longrightarrow Q\mathbb{E}$ be a view of $\mathbb{E}$. Suppose $t : T \rightarrowtail T'$ consists of two database states for $\mathbf{V}$ and a database state monomorphism, with T' being an insert update of T and with $T = V^*D$ for some database state D of $\mathbb{E}$. We say that the insert t is *propagatable* when there exists an initial $m : D \rightarrowtail D'$ among all those database states D'' with $m' : D \rightarrowtail D''$ for which $V^*D'' = T'$ and $V^*m' = t$. 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 $D'' \rightarrowtail D$).

Since a view is just a database state, we know how to insert or delete instances. Intuitively a specified view insert/delete is then propagatable if there is a unique "minimal" insert/delete on the underlying information system whose restriction to the view (via V^*) is the given view insert/delete.

Notice that propagatability (view updatability) is in principle dependent on the database state (the model of the specification) which is being updated — we have defined when an insert or delete of a view (database state) is propagatable, rather than trying to determine for which view schemata inserts and deletes can always be propagated. In fact we can often prove that for given 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 some propositions analysing this.

It is important to note that 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) model of the underlying information system among those models that could achieve the update of the view we say that the view is updatable. The only invalid view updates are those which are in fact impossible to achieve in the underlying information system, or those which could be derived from multiple non-isomorphic minimal or maximal models of the underlying information system.

5 Schema updatability

Definition 12 A view $V : \mathbf{V} \longrightarrow Q\mathbb{E}$ is called *insert* (respectively *delete*) *updatable* at an entity $W \in \mathbf{V}$ when all inserts (respectively deletes) into (respectively from) W are propagatable, independently of the database state. (Note that an insert or delete *at* W changes the database state's value only at W — the values in the model of other entities and attributes remain unchanged.)

In this section we establish insert or delete updatability at $W \in \mathbf{V}$ (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.

To establish notation, assume that $V : \mathbf{V} \longrightarrow Q\mathbb{E}$ is a sketch morphism, that T and T' are models of $\mathbf{V}$, that T' is an insert or delete update of T at W, and that D, D' and D'' are models of $\mathbb{E}$. Suppose further that $T = V^*D$ and that $T' = V^*D' = V^*D''$. 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 $V^*m = V^*m' = t$.

In most of the following propositions we will suppose for simplicity that $\mathbb{E}$ has no cones except the empty cone with vertex 1, and no cocones except attribute cocones. With care the propositions can be generalised to sketches $\mathbb{E}$ which do not meet this restriction, provided that W is not in any of the cones or cocones except perhaps as specified explicitly in the hypotheses. Similarly we will assume for simplicity that V is an injective sketch morphism. We begin by considering cases where V is just a view of a part of $\mathbb{E}$.

Proposition 13 *Suppose $VW \in \mathbb{E}$. If VW is not the initial node in any commutative diagram in $\mathbb{E}$ and all of the arrows out of VW in $\mathbb{E}$ are in the image of V, then V is insert updatable at W. Conversely, 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.

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, $D'gD'f = D'gt_W Df = DgDf$. Furthermore $m : D \rightarrowtail D'$ is initial among the appropriate $m' : D \rightarrowtail D''$ since it is initial at each component.

Proposition 13 says that a view which can "see enough" is updatable. For example, if the view were to include Medical practitioner, Practice agreement, and Hospital, along with the two arrows between them (see Figure 1), then the view is insert updatable, but not delete updatable, at Practice agreement.

In many of the following propositions W is assumed to be the only entity in the view, and $\mathbb{E}$ will be very simple. This might seem rather restrictive. In fact, the single entity view is in accord with common practice where views are frequently required to be the result of a single query, so the view should be a single object $\{W\} = \mathbf{V}$ with its image in $Q\mathbb{E}$. In our applications we encourage larger structured $\mathbf{V}$, but the following propositions are nevertheless useful then because we can search for parts of $\mathbf{V}$ which match the premises of the propositions and either find a counterexample to updatability at W, or partition $\mathbf{V} - \{W\}$ and argue that updatability for each partition as a view, whether concurrently or serially, implies the updatability of $\mathbf{V}$ at W. Similarly the propositions can be applied to large complex $\mathbb{E}$ because updatability is a "local" phenomenon: Inserts or deletes at W will be updatable according as to whether they are updatable in the restriction of $\mathbb{E}$ to objects "near" W.

Proposition 14 *Suppose that $\mathbf{V} = \{W\}$, 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 of at least two elements. Then V is not insert updatable at W.*

Proof. Choose two distinct elements $a, b : 1 \longrightarrow A$. If the insert is non-trivial and atomic then there is an element $w : 1 \longrightarrow T'W$ which is not in TW and $T'W = TW + \{w\}$. Consider D' and D'' defined by $D'W = D''W = T'W$, and of course $D'A = D''A = T'A = TA$ (attributes are constant for all models), with $D'fw = a$ and $D''fw = b$ and $D'f = D''f = Tf$ when restricted to TW. But now $D \longrightarrow D'$ and $D \longrightarrow D''$ are incomparable but minimal so there is no initial object and the view update is not propagatable.

Thus we should require that W has all of its attributes in its view. For simplicity we will in fact assume that W has no attributes for the remainder of this section, but the propositions can be generalised to arbitrary W provided all of the attributes of W do appear in $\mathbf{V}$.

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

Proposition 15 *Suppose that $\mathbf{V} = \{W\}$, and $\mathbb{E}$ has as graph $f : B \longrightarrow A$ where A has an element $a : 1 \longrightarrow A$. Let VW be the pullback*

$$\begin{tikzcd} VW \arrow[r] \arrow[d] & B \arrow[d, "f"] \\ 1 \arrow[r, "a"] & A \end{tikzcd}$$

Then V is insert updatable at W.

Proof. Write $T'W = TW + W_0$ with t the inclusion of the first summand, which we can do since T' is an insert update of T (writing $+$ for disjoint union in $\mathbf{Set}_0$). Let $D'B = DB + W_0$, and $D'A = DA$, and 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, and, calculating the pullback in $\mathbf{Set}$, $V^*D' = T'$ and $V^*m = t$. Suppose $m' : D \rightarrowtail D''$ is another such model. Then there is a unique natural transformation $i : D' \longrightarrow D''$ commuting with m and m' since with $D''1 = 1$ and with DA and DB fixed inside $D''A$ and $D''B$, $D''f$ must have as fibre over a, $(Df)^{-1}(a) + W_0$ in order for the pullback to be $T'W$, and these fully determine the components on i. Thus, V is insert updatable.

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 14. But the fact that VW arises as a pullback determines the values that the function must take, and that all those values must be fibred over a.

Proposition 15 is also important because it is an example of an update that many view systems would prohibit [13] despite its practical importance. As an example which arises naturally consider a view of Figure 1 which arises from choosing a particular specialisation. This is the view used by an administrator of a particular college, and it should be updatable.

If the hypotheses of Proposition 15 were generalised to replace 1 by an entity C the proposition would no longer hold. However, if C is included in the view along with the pullback VW and the arrow between them then we recover insert updatability.

Alternatively, the hypotheses can be generalised 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.

Proposition 16 *Suppose that* $\mathbf{V} = \{W\}$*, and* $\mathbb{E}$ *has two entities* A *and* B*. Let* VW *be the coproduct of* A *and* B*. Then* V *is not insert updatable.*

Proof. (Sketch:) The two models D' and D'' obtained by adding the set difference $T'W - TW$ to A and B respectively are incomparable and minimal.

Proposition 17 *Suppose that* $\mathbf{V} = \{A_0 \longrightarrow W\}$*, and* $\mathbb{E}$ *has two entities* A *and* B*. Let* $VA_0 = A$ *and let* VW *be the coproduct of* A *and* B*. Then* V *is insert updatable.*

Proof. (Sketch:) In contrast to the proof of the previous proposition, this time an element of $T'W - TW$ corresponds to an element of $T'A_0$ or 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 noted 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 D' so constructed is initial, and the view is insert updatable at W.

Proposition 18 *Suppose that* $\mathbf{V} = \{W\}$*, 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, in which case the view insert is not propagatable.

There are many more results of similar interest and considerable utility. This section has provided a sample of results indicating a range of circumstances that can easily be dealt with.

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

6 Related work

In the last decade there has been considerable growth in the use of sketches to support data modelling. Among this work Piessens has obtained results on the algorithmic determination of equivalences of model categories [27] which were intended to support plans for view integration. Meanwhile Diskin and Cadish have used sketches for a variety of modelling purposes including for example [14] and [15]. They have been concentrating on developing the diagrammatic language of "diagram operations". Others, including Lippe and ter Hofstede [25], Islam and Phoa [17], and Baklawski et al [4], have been using category theory for data modelling.

Recent work on updating problems has included work by 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 $D \rightarrowtail D''$) 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 [24].

Meanwhile, the authors have recently been futher testing the techniques presented here. Johnson and Dampney [9] have used the techniques in a case study; Dampney, Johnson and Rosebrugh [12] explore the implications for semantic data modelling and present a simplified form of the techniques to the database community; and Johnson and Rosebrugh [20] show how the techniques can be used for database interoperability for computer supported cooperative work. Johnson, Rosebrugh and Wood [21] have developed a new mathematical foundation that unifies the treatment of specifications, updates, and model categories. And in current work the present authors are exploring the relationship between our approach to the view update problem and the frame problem in software engineering [18], [5].

7 Conclusion

After defining the sketch data model in Section 2 we defined views (Section 3) in a way that ensures that the view structure is itself a sketch data model, and that offers maximum generality in the sense that the view can be constructed from any data that can be obtained from queries of the underlying database. In this framework we have proposed a new solution to the view update problem (Section 4), and shown in Section 5 how we can still obtain results about the updatability of schemata.

The work presented here has a number of limitations:

1. Views take values in $Q\!\!E$ which contains all structural queries, but no arithmetic queries that could summarise, rather than extract and manipulate, data.
2. The updates dealt with are only insert and delete updates. We don't yet treat modifications of data in situ.
3. We provide no special treatment of nulls (in agreement with, for example, Date's recommendation [13] that systems should not support nulls).
4. We have not given detailed consideration to implementational issues. In particular the treatment of both the *when* and the *how* of view updating by universal properties does not directly address implementational issues (but see the remarks below on computational category theory).

Each of these is the subject of ongoing current research.

Despite the limitations, the new approach to views has significant advantages:

1. The sketch data model has been extended to incorporate views, and the extension is very general allowing data based on any structural query to be viewed in any sketch data model schema, subject only to the compatibility with the underlying information system implied by V being a sketch morphism (and this last is as we would expect — we can't constrain the data in the view more than it is constrained in the underlying information system since the former is derived from the latter).
2. View updatability is defined once and for all in a single consistent framework based on a universal property among models. Arguably the universal property gives the most general reasonable definition to view updatability possible.
3. The framework presented here links well with computational category theory work being carried out in Italy, the UK, Canada and Australia. That work has developed repositories, graphical tools, and elementary algorithms, that amount to a rapid prototyping tool for systems specified using the sketch data model.
4. The "closedness" obtained by having a view be itself a sketch data model allows views of views etc. It also supports well proposals for using views as the interface for database interoperability and for federated information systems.

5. The propositions presented in section 5 and similar propositions allow us to work with schema updatability (rather than model based updatability) in the usual way, and the proofs of the propositions embody the code required to carry out the update without resorting to general universal property algorithms.

These developments have depended fundamentally on using a formal methods framework, rather than the more usual semi-formal methodologies, and this led to the universal property being based on models rather than the more usual schema based definitions.

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. *TODS*, 17:718–743, 1992.
3. M. Barr and C. Wells. *Category theory for computing science*. Prentice-Hall, second edition, 1995.
4. K. Baklawski, D. Dimovici and W. White. A categorical approach to database semantics. *Mathematical Structures in Computer Science*, 4:147–183, 1994.
5. A. Borgida, J. Mylopoulos and R. Reiter. And Nothing Else Changes: The Frame Problem in Procedure Specifications. *Proceedings of the Fifteenth International Conference on Software Engineering*, IEEE Computer Society Press, 1993.
6. P. P. -S. Chen. The Entity-Relationship Model—Toward a Unified View of Data. *TODS*, 2:9–36, 1976.
7. C. N. G. Dampney and Michael Johnson. TIME Compliant Corporate Data Model Validation. Consultants' report to Telecom Australia, 1991.
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 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.
10. 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.
11. 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.
12. C. N. G. Dampney, Michael Johnson, and Robert Rosebrugh. View Updates in a Semantic Data Model Paradigm. Proceedings of ADC, IEEE Computer Society, in press, 2001.
13. C. J. Date. *Introduction to Database Systems, Volume 2*. Addison-Wesley, 1983.
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*, 2000.

16. H. Ehrig and B. Mahr. *Fundamentals of algebraic specifications*. Springer-Verlag, 1985.
17. 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.
18. D. Jackson. Structuring Z Specifications with Views. *ACM Transactions on Software Engineering and Methodology*, 4:365–389, 1995.
19. 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.
20. Michael Johnson and Robert Rosebrugh. Database interoperability through state based logical data independence. *Proceedings of the Fifth International Conference on Computer Supported Cooperative Work in Design*, IEEE Hong Kong, 161–166, 2000.
21. Michael Johnson, Robert Rosebrugh, and R. J. Wood. Entity-relationship models and sketches. Submitted to *Theory and Applications of Categories*, 2001.
22. Rom Langerak. View updates in relational databases with an independent scheme. *TODS*, 15:40–66, 1990.
23. A. Y. Levy, A. O. Mendelzon, D. Srivastava, Y. Sagiv. Answering queries using views. *ACM PODS-95*, 1995.
24. C. Lecluse and N. Spyratos. Implementing queries and updates on universal scheme interfaces. *VLDB*, 62–75, 1988.
25. E. Lippe and A ter Hofstede. A category theoretical approach to conceptual data modelling. *RAIRO Theoretical Informatics and Applications*, 30:31–79, 1996.
26. Saunders Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics 5, Springer Verlag, 1971.
27. F. Piessens and Eric Steegmans. Selective Attribute Elimination for Categorical Data Specifications. Proceedings of the 6th International AMAST. Ed. Michael Johnson. *Lecture Notes in Computer Science*, 1349:424-436, 1997.
28. J. D. Ullman. *Principles of Database and Knowledge-Base Systems*. Volume 1, Computer Science Press, 1988.
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.