Update Algorithms for the Sketch Data Model

Michael Johnson
Macquarie University, Australia Email: mike@ics.mq.edu.au

Robert Rosebrugh
Mount Allison University, Canada Email: rrosebru@mta.ca

Abstract

The authors have developed a new approach to database interoperability using the sketch data model. *That technique has now been used in a number of applications, but an important question remains: What are the algorithms that support updates in the sketch data model? The question has significant content since the sketch data model uses* EA-sketches *to specify data structures, and these include constraint and other information not normally supported by relational database management systems. In this paper we answer the question by using the framework of EA sketches to give a detailed mathematical treatment of database updates, providing a formal definition of insert update together with an algorithm which provably achieves updates. The algorithm is new as it is the first to treat data and constraints on an equal categorical footing. We also note that further exactness properties (limits and colimits) can aid specification, and we provide algorithms for updates of EA sketched databases with finite limits. These are the first update algorithms for such databases. The sketch data model is being used in industry for designing interoperations for computer supported cooperative workand computer assisted software engineering (CASE) tools are under development. The paper is predominantly theoretical, and provides an important link needed for CASE tool development.*

Keywords: Category theory, data model, mathematical specification, database interoperation

1. Introduction

For a number of years the authors, together with others, have been developing the theoretical foundations for a new semantic data modelling technique — the *Sketch Data Model* (SDM). Meanwhile, Dampney and Johnson have been applying the insights gained from the theoretical work to data modelling problems in consultancies with major Australian Companies [8], [28], [10], [7].

At CSCWD2000 the authors presented a new technique for supporting database interoperability based on the sketch data model [20]. The theoretical underpinnings of that technique, which depend on the solution of the view update problem, have since been further developed [22] and [21], and the technique has been used successfully in industry [9]. Furthermore, the technique was extended to *half-duplex interoperations* once the need for that extension became apparent in industrial applications [19], and these half-duplex interoperations are being used now to support interoperations on semi-structured databases used for design.

However, there remains a fundamental question which has not, until now, been addressed. While it is possible to use the relational data model to implement database interoperations which have been designed using the sketch data model, what are the update algorithms which would be the basis of a sketch data model database management system? This paper provides a first answer to that question, by developing a formal treatment of updates along with algorithms that can be proved to be compliant with that treatment.

The sketch data model (defined formally below) can be viewed as an extension of the widely used Entity-Relationship (ER) approach [5] to information modelling. In addition to the graphical advantages of the ER approach, the SDM incorporates many of the insights of categorical universal algebra [3] (chapter 4), [24], allowing a much more detailed specification of constraints, and a commensurate improvement in the quality of the specification and design process. Thus, the principal successes of the SDM approach have been in the early design stage. If the design is intended to be implemented, the model is usually just viewed as an ER model (forgetting the detailed constraint specifications) and then it is implemented as a relational database in the usual way.

The problem addressed by this paper is to develop the theory to bridge the gap between the SDM based design phase and implementations, while taking into account the full set of constraint specifications.

We need not concern ourselves with the details of the data model used for the implementation, which might be the common relational model [6], or even more appropriately the functional model [13] (Section 5.4), since for constraints the critical issue is the update process. Thus we seek (and in Section 4 provide) a formal definition of elementary update in the SDM framework, and algorithms which can be proven to achieve such updates (Section 5).

Before proceeding we briefly review the early results in the previous theoretical work on the SDM, and the work of some other authors who have also proposed sketches as a basis for the study of semantic data models.

Dampney, Johnson and Monro [11] were the first to note that the categorical universal algebra approach implied that the classifying category for a data specification included objects representing all of the query results for the corresponding database. This important fact gives the SDM a categorical closure akin to relational completeness: Any of a class of categorical operations (which includes the standard relational operations) can be applied to objects in the classifying category and the results will always be objects of the classifying category. Dampney et al also recognised the importance of update algorithms, but their approach to updating was clumsy and incomplete. Dampney and Johnson [18] demonstrated the importance of commutative diagrams in constraint specification (taking advantage of the query objects), and gave a range of examples indicating the breadth of the constraints that can be dealt with. Johnson [16] began the development of the logic for the query objects and [17] provides a detailed development of the categorical logic. Johnson, Rosebrugh and Wood [23] provide a theoretical treatment of updates in terms of the span construction, and prove the important technical result that a span of SDM models in a lextensive category $\mathcal{C}$ is an ordinary SDM model in $\mathbf{spn}\,\mathcal{C}$.

Meanwhile, Piessens [26, 27] used sketches as part of his notion of 'data specification'. Motivated by the problem of view integration he has obtained results on the algorithmic determination of equivalences of model categories for certain classes of data specifications. Diskin and Cadish [14], [15] describe enhancements to entity-relationship models also using sketches. Their perspective differs from ours in using 'diagram operations' on sketches to model queries.

The remainder of this paper is arranged as follows. In Section 2 we review the category theoretic view we take of ER models and of semantic data modelling in general. In Section 3 we present the definition and develop the elementary properties of EA sketches — the basis of the SDM. Section 4 introduces the theory of insert updates (delete updates can be analysed in a dual manner). That section, and Section 5, form the core of the paper as they develop the mathematical theory of an update in the SDM framework, and demonstrate the connection between that theory and the algorithms presented in Section 5.

2. Background

This section contains a brief review of our categorical view of ER models. For a fuller description see [18] or [20].

An *entity* is a set about which a database owner has information. For example a university might include STUDENT, UNIT, LECTURER, ... as entities in its information model. Entities have certain *attributes* or properties — for example students have a name, address, degree program and so on. Among the attributes of an entity there may be a specified *key attribute* — a student identification number would be an example. A *relationship* among entities is a (many-many) relation — for example students are enrolled in units. Typically there are many students in any one unit, and also any one student is typically enrolled in several units.

An ER model is a set of entities, together with their attributes, and any relationships among them. Notice that the model records the structure of the information, but not yet any instances — no data. In essence such a model is a structured datatype specification, or in mathematical terms a specification for a variety of (flat, multi-sorted) algebras.

ER models are generally presented graphically, but with a range of notations and extensions. We will use a specific directed multigraph (hereafter such graphs will simply be called "graphs") which can be obtained from each ER model as follows. Given an ER model, the nodes of its graph G will be the disjoint union of the entities, their attributes, the relationships and a special node denoted 1. The (directed) edges of G will consist of an edge from each entity to each of its attributes; an edge from each relation to each of the entities that it relates; and for each attribute A with domain having n elements (that is there are n different values for that attribute), n distinct edges $1 \longrightarrow A$.

Notice that in this graph entities and relationships are treated similarly. Henceforward we drop the distinction between entities and relationships. This point of view is also espoused by other writers on database theory, for example C. J. Date, [12]. The name EA sketch emphasises the technical importance of the distinction between entities and attributes, first noted in [18].

3. EA Sketches

In this section we introduce EA sketches, the fundamental notion of the sketch data model. To establish our notation we begin by briefly describing the language of sketches and their models. For fuller treatment we refer the reader to [1] or [2].

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 (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 1 *A sketch $\mathbb{E}$ is a 4-tuple $\mathbb{E} = (G, \mathbf{D}, \mathcal{L}, \mathcal{C})$ where G is a directed graph, $\mathbf{D}$ is a set of pairs of paths in G with common source and target (the commutative diagrams) and $\mathcal{L}$ (respectively $\mathcal{C}$) is a set of cones (respectively cocones) in G.*

Definition 2 *A* model *M of a sketch $\mathbb{E}$ in a category $\mathbf{S}$ is a coherent 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).*

For a sketch $\mathbb{E}$ we denote the category generated by G subject to the relations $\mathbf{D}$ by $\widetilde{\mathbb{E}}$. Using the evident inclusion $G \longrightarrow \widetilde{\mathbb{E}}$ 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 $\widetilde{\mathbb{E}}$.

A model M of $\mathbb{E}$ in $\mathbf{S}$ extends to a functor $\widetilde{M} : \widetilde{\mathbb{E}} \longrightarrow \mathbf{S}$ which takes every (co)cone in $\mathcal{L}$ ($\mathcal{C}$) to a (co)limit (co)cone. If M and M' are models a *homomorphism* $\phi : M \longrightarrow M'$ is simply 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 $\mathrm{Mod}(\mathbb{E}, \mathbf{S})$, 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, (finite-product, $\emptyset$) sketches correspond to (multi-sorted) algebraic theories.

Definition 3 *An* EA 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 coproduct (discrete) cocones whose injections are elements are called* attributes. *An attribute is not the domain of an arrow.*
- *The underlying graph of $\mathbb{E}$ is acyclic and finite.*

A node which is neither an attribute, nor 1, is called an entity. *An EA sketch is* keyed *if each entity E has a specified monic arrow $k_E : E \rightarrowtail A_E$ to some attribute A_E. An entity is called a* base entity *if it is not the vertex of any cone nor in the base of any cocone of $\mathbb{E}$. A* finite limit EA sketch *is an EA sketch with empty $\mathcal{C}$.*

Remark 4 i) Recall that in general an arrow m is *monic* if $mf = mg$ implies $f = g$, and that in **Set** such an arrow is an injective function. We say that an arrow m in an EA sketch is monic when its image in every model is monic. This can be assured in a number of ways. Choosing one for definiteness, we call m monic in $\mathbb{E}$, and denote it $m : X \rightarrowtail Y$, when there is a cone in $\mathcal{L}$ whose base has image $X \rightarrowtail Y \leftarrowtail X$ where both arrows are m, and whose two projections onto X are equal.
ii) The base entities correspond to the *strong entities* in ERA models, while an entity with a monic arrow to a base entity or which is in the base of a cocone with vertex a base entity corresponds to a *weak entity*. In practice we expect that when the EA sketch is keyed, the key for a weak entity will be the composite of the key for its base entity and the inclusion.
iii) Notice that every ER model yields an EA sketch: Let G be the corresponding graph as described in Section 2, let $\mathbf{D}$ be empty and let $\mathcal{L}$ contain only the empty cone with vertex 1. Let $\mathcal{C}$ be the set of discrete cocones of elements of each attribute. If we wish to ensure that the relations can only be modelled as true relations, add for each 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. If the ER model is extended to include indications of key attributes add cones to $\mathcal{L}$ to ensure that the corresponding arrows to those attributes are monic.
iv) Now the extra power of the sketch data model is clear: $\mathbf{D}$ can be used to record constraints, and $\mathcal{L}$ and $\mathcal{C}$ can be used to calculate query results from other objects, and these query results can in turn be used to add constraints, etc. Furthermore, all this takes place with a firm and well-understood mathematical foundation developed for universal algebra.

The appropriate receiving categories for models of EA sketches (and hence for their databases' states) should have finite limits and finite coproducts. The coproducts will be assumed to be disjoint and universal. Such categories are called *lextensive* and have already been studied [4]. They are examples of *distributive categories*, and they have many applications in theoretical computer science [29]. For the record,

Definition 5 *A category $\mathbf{S}$ is* lextensive*, if it has finite limits and finite coproducts which are disjoint and universal.*

For the balance of this article, $\mathbf{S}$ will be a lextensive category. We note that the definition of lextensive in [4] differs from that above, though the definition there is shown to be equivalent to Definition 5.

Remark 6 Notice that the definition of EA-sketch does not rule out the possibility of an *inconsistent sketch*, i. e. a sketch with no non-trivial models. For example, the sketch might include attributes A_2 and A_3 with 2 and 3 elements respectively and then specify that their sum is 1 which implies that the sketch has no models in $\mathbf{S}$ unless $\mathbf{S}$ is degenerate, that is $0 \cong 1$ in $\mathbf{S}$.

Definition 7 *A* database state *D in $\mathbf{S}$ for an EA sketch $\mathbb{E}$ is a model of $\mathbb{E}$ in $\mathbf{S}$. The* category of database states of $\mathbb{E}$ in $\mathbf{S}$ *is the category of models* $\mathrm{Mod}(\mathbb{E}, \mathbf{S})$ *of $\mathbb{E}$ in $\mathbf{S}$. For brevity we sometimes denote it by* $\mathrm{Mod}(\mathbb{E})$ *when $\mathbf{S}$ is the category of finite sets,* $\mathbf{Set}_0$

4. Insert updates

Under what circumstances can we say, given two database states for the same EA sketch, that one is an *update* of the other? The question is important, and requires a mathematical answer, if we are to prove that update algorithms have properties corresponding to consistency and completeness.

Of course, in most databases, any state can be obtained as an update of any other state: We can for example delete all the data from the first state, and then insert all the data for the second. So, we restrict our attention here to *elementary updates* — those that can be obtained via a single instance insertion or deletion.

As is common practice, we will not deal theoretically with updates which modify existing values. Instead, such modifications are expected to be obtained by (elementary) deletions and insertions. Furthermore, a formalism for elementary deletions can be obtained from our treatment of elementary insertions. Thus, we focus our attention on elementary insertions.

Finally, a few words about *cascades*. There remains some controversy in the database community about whether elementary updates should cascade. (In the following it is convenient to speak about entities and relations even though relations are themselves entities in our framework — whenever $X \longrightarrow Y$ we can say that X is a relation involving the entity Y.) Suppose we attempt to delete an instance of an entity, which is related to another instance of another entity. Some systems will prohibit this operation, requiring the relation instance to be deleted first, while other systems will cascade the delete, deleting the relation instance (and any relation instances that it takes part in etc.) automatically. Similarly insertions may cascade (an insertion of a relation instance may automatically lead to the insertion of a new entity instance to be related by the relation instance, etc.) or the system may prohibit cascades and require that all instances to be related are inserted before the relation instance can be inserted.

Now to the formal treatment of insertions. We will insert a new instance x in a database state D at a given entity E.

Let $\mathbb{E}$ be an EA sketch, and E be an object of $\mathbb{E}$. We define $\mathbb{E}[x_E]$ to be $\mathbb{E}$ augmented with a new arrow $x : 1 \longrightarrow E$ (and no new commutative diagrams, cones or cocones). Write J for the evident inclusion $\widetilde{\mathbb{E}} \longrightarrow \widetilde{\mathbb{E}[x_E]}$.

Write $\mathrm{Mod}(\mathbb{E})$ for the category of database states. Thus objects of $\mathrm{Mod}(\mathbb{E})$ are certain functors $D : \widetilde{\mathbb{E}} \longrightarrow \mathbf{Set}_0$. Write, abusing notation, $U = J^*$ for the functor $[\widetilde{\mathbb{E}[x_E]}, \mathbf{Set}_0] \longrightarrow [\widetilde{\mathbb{E}}, \mathbf{Set}_0]$ given by composition with J, and for its restriction $U : \mathrm{Mod}(\mathbb{E}[x_E]) \longrightarrow \mathrm{Mod}(\mathbb{E})$. Now J^* has a left adjoint computed as the left Kan extension of the functor along the inclusion $J : \widetilde{\mathbb{E}} \longrightarrow \widetilde{\mathbb{E}[x_E]}$ and in the case of finite limit EA sketches, the adjoint restricts to the category of database states. Call the left adjoint L and let η be the unit of the adjunction.

Definition 8 *Suppose D is an $\mathbb{E}$ database state, E an object of $\mathbb{E}$, and D' an $\mathbb{E}[x_E]$ database state. Then $m : D \rightarrowtail UD'$ is an* insert update *of D at E if there exists an epi $\phi : LD \longrightarrow D'$ such that $(U\phi)\eta_D = m$.*

Remark 9 i) Intuitively, LD should be thought of as the universal database state containing D and a new constant x of type E. The definition requires D' to be "squeezed" between D and LD, coherently with the way D is included in both D' and LD.
ii) Our definition permits, but does not mandate, cascades. The Kan extension, LD, cascades maximally. Requiring D' to be an epimorphic image of LD means that what in the Kan extension is a cascaded insert, may in D' be a relation instance assignment to an already extant entity instance or attribute value.
iii) In fact, LD will not in general be a database state if the sketch includes coproduct cocones, and it will certainly not be a database state if the sketch is keyed. This is easy to see by using the Kan extension formula [25] (page 236) to calculate the value of $LD(A_E)$, where A_E is the key attribute for E. The "maximal cascade" mentioned above corresponds to $LD(A_E) = D(A_E) + 1$, and so LD can't be a database state. Nevertheless, D' is a database state, and ϕ indicates how cascaded inserts which have disrupted coproducts in LD are replaced by assignments to extant instances or values in D'.
iv) Elementary properties of adjunctions ensure that if ϕ, as in the definition, exists, then it is unique.

5. The algorithms

We provide pseudocode algorithms for the two most important cases of insert updates, linear and finite-limit EA sketches. In each case we remark briefly on some of the finer points of the algorithms, and in the linear case we establish in detail the connection between the theory (Section 4) and the algorithms.

In the following sections we write $A + B$ for the disjoint union of A and B when A and B are sets. We remind the reader that we compose functions from the right, thus $pqa = p(q(a))$ means evaluate q at a, and then apply p to the result.

5.1 The linear case

A sketch is called *linear* when its sets of cones and cocones are both empty. Of course this can never occur with EA sketches which are required to have a specified cone (Definition 3). The corresponding notion for an EA sketch is

Definition 10 *An EA sketch $\mathbb{E} = (G, \mathbf{D}, \mathcal{L}, \mathcal{C})$ is called* linear *when $\mathcal{C}$ is empty and $\mathcal{L}$ contains just a single cone (which must then have an empty base).*

We provide an insert update algorithm (Figure 1) for the linear case. The algorithm forms the basis for algorithms which deal with further exactness properties (limits and colimits) and is useful for abstract data specifications which typically record no attributes. Because it is so fundamental, we will, after a brief discussion of some of its facets, establish in detail some of its properties, and its relationship with the theory presented in Section 4

The algorithm deals with a database state `D`, and constructs a new database state `D'`. It thus requires storage for sets, which appear in the algorithm as, for example, `D'(E)`, and for functions, which are presumed to be stored as their graphs (ordered pairs of (domain, codomain) values). As an example of a function, the line `D'(x_E)1 = t` says that the pair $(1, t)$ should be added to the graph of the function corresponding to x_E in the database (partial) state `D'`.

The algorithm begins by adding the element `t` and then prompts the user for all the images of the new `t`. The user's proposed images are checked to ensure that commutative diagrams are preserved (and this needs to be done carefully, because at this point many of the functions are only partial functions). The user is also given an abort option. This is necessary since it is possible to begin an insert which can never be completed without violating commutative diagram constraints.

We now explore some of the mathematical properties of the algorithm.

Proposition 11 *If the procedure* `INS` *returns with* `ins = TRUE`*, then* `D'` *is a database state for $\mathbb{E}[x_E]$.*

Proof. Suppose `INS` returns with `ins = TRUE`.

To show that D' is a database state for $\mathbb{E}[x_E]$ we check explicitly that D' is a graph morphism into the underlying graph of **Set** and that the required diagrams commute.

Firstly notice $D'(1) = D(1) = 1$, since E being an entity is not 1 (Definition 3). Note the abuse of notation: As is customary, we write 1 for the vertex of the empty cone, for a set with one element, and as the name of that element. Next notice from statement 3 of `INS` that $t = D'(x_E)1$ (and by acyclicity, this is never modified). Thus $D'(x_E)$ is a total function.

The node part of the graph morphism is straightforward: For all E', $D'(E')$ is a set, defined by D unless $E' = E$ whence $D'(E') = D(E) + \{t\}$.

The arrow part of the graph morphism follows since for all $a : E'' \longrightarrow E'$, $D'(a)$ is a total function: If E'' is not E and a is not x_E, then $D'(a) = D(a)$. If $a = x_E$, $D'(a)$ is the total function defined above. The only remaining case is $a : E \longrightarrow E'$. For such a, the required mapping is defined by $D(a)$ on $D(E)$ by the first statement of `INS`, so it remains to see that $D(a)t$ is defined. Since `ins` returns `TRUE`, but `ins` is set `FALSE` at the beginning of `INTER_ASSIGN`, either there are no a's to check in the loop in `INS`, so we are done, or $D'(a)t$ must have been assigned in `EXTEND_FUNCTION`. Indeed, `INTER_ASSIGN` had to return `ins = TRUE` on every loop iteration (otherwise we exit with `ins FALSE`, contrary to assumption). For `INTER_ASSIGN` to return `ins = TRUE`, `EXTEND_FUNCTION` returned `ins = TRUE` (since `ins` is initialized to `FALSE` in `INTER_ASSIGN`). `EXTEND_FUNCTION` only assigns `ins = TRUE` after assignment of $D'(a)t$.

Finally, all commutative diagrams in $\mathbb{E}[x_E]$ are respected by D': Any commutative diagram not involving E is unchanged. If a commutative diagram involving E does not start from E, then in the diagram E is in the codomain of at least one arrow b not equal to x_E, but for each such b, t is not in the image of $D'(b)$ since $D'(b) = D(b)$, so the diagram still commutes. So suppose we have a commutative diagram starting at E, say $(qb, q'b')$, with q, q' (possibly empty) paths. Suppose the diagram fails to commute in D'. This can only be because $D'(qb)t$ is not equal to $D'(q'b')t$. Since `INS` returned `ins = TRUE`, `INTER_ASSIGN` must have returned `ins = TRUE` on each iteration of the main loop of `INS`. Thus `EXTEND_FUNCTION` must have set `ins = TRUE` (for each call to `INTER_ASSIGN`). Hence, `CHCD` evaluated to `TRUE` when called with b and when called with b' (each as second parameter). But then if $b' = b$, the first `if` in `CHCD` found $D'(qb)t$ not equal to $D'(q'b)t$, and so set `CHCD` to `FALSE`, contradicting the assumption that `INS` returned `TRUE`. On the other hand, if b is not equal to b', without loss of generality, suppose the call to `CHCD` with b was the latter, then $D'(b')t$ was already defined when this call occurred, so the second `if` in `CHCD` found $D'(qb)t$ not equal to $D'(q'b')t$, and so set `CHCD` to `FALSE`, contradicting the assumption that `INS` returned `TRUE`. ∎

For the next proposition, we will need an explicit description of LD.

Lemma 12 *For E' not equal to E*

$$LD(E') = D(E') + \{\beta : E \longrightarrow E'\}$$

```
/* INSERT ALGORITHM FOR LINEAR EA SKETCHES

Algorithm INS(t, D, E, ins, D’)
/*  pre: D a database state, E an object of D, and t not in D(E).
/* post: if ins is FALSE, D is unchanged, and D’ is garbage;
/*       if ins is TRUE t is in D’(E), and D’ is a database state for
/*       the extended sketch E[x].

D’ = D                   /* D’ will be the new database state
D’(E) = D’(E) + { t }    /* This is disjoint union
D’(x_E)1 = t             /* Abuse notation writing 1 for the element of 1
ins = TRUE
for each a : E --> E’
/* loop completion means D’(a)t is validly defined for each arrow a ie insert successful
   INTER_ASSIGN(D’, a, t, ins)
   /* Let user choose D’(a)t ’s value. Does not cascade (allow new element creation)
   if not ins
        exit   /* If INTER_ASSIGN is aborted we exit with ins FALSE

/* end of INS

Algorithm INTER_ASSIGN(D’, a, t, ins)
/* Interactively asks user for a valid value of D’(a)t

ins = FALSE
repeat
   Output(’type value of D’(a)t in D’(E’) or CTRL-Q to quit’)
   Input(t’)
   if t’ not in D’(E’)
      output( ‘t’ not in D’(E’); try again’)
      break
   if t’ = CTRL-Q
      exit    /* Needed since it’s possible to attempt an insert
              /* which would never satisfy CD requirements.
   EXTEND_FUNCTION(D’, a, t, t’, ins)
until ins = TRUE                    /* so user must have succeeded
/* end of INTER_ASSIGN

Algorithm EXTEND_FUNCTION(D’,a, t, t’, ins)
/* Checks cd’s and extends D’(a) by t |--> t’ if ok.

if CHCD(D’, a, t, t’)
   D’(a)t = t’
   ins = TRUE
else
   Output(’CD violation, extend function failed’)
   ins = FALSE  /* assigned for emphasis, ins must be false here
/* end of EXTEND_FUNCTION

Algorithm CHCD(D’, a, t, t’) : Boolean
/* Returns TRUE if the definition D’(a)t = t’ respects commutative diagrams.
/* We check separately CD’s with both paths beginning a, since D’(a)t not yet defined
/* (such paths would be checked twice, but second if is false).

  CHCD = TRUE
  for each CD = (pa, p’a)     /* p, p’ are paths; the cd paths both begin with a
       if not ( D’(p)t’ = D’(p’)t’ )
          /* Execute if test for comm fails
          CHCD = FALSE
  for each CD = (pa, p’) or (p’, pa)   /* p, p’ are paths
       if not ( D’(p’)t = undef ) and not ( D’(p)t’ = D’(p’)t )
          /* Execute if we can test for comm and the test fails.
          /* If we can’t yet test (D’(p’)t undef) we’ll test this when defining D’(p’)t.
          CHCD = FALSE

/* end of CHCD
```

Figure 1. An insert update algorithm for linear EA sketches

Proof. $LD(E') = \operatorname{colim}(J/E' \longrightarrow \widetilde{\mathbb{E}} \longrightarrow \mathbf{Set}_0)$ where the first arrow is the projection from the comma category, and the second is the database state D. Since J is bijective on objects, each object E' of $\widetilde{\mathbb{E}}[x_E]$ is of the form JF for some F. Consider J/JF, whose objects are of the form $(A, \alpha : JA \longrightarrow JF)$. Such α are either Ja for $a : A \longrightarrow F$, or $\beta x : J1 \longrightarrow JF$ with $\beta = Jb$, some b. Notice that there are no morphisms in J/JF to or from objects of the second kind (because there are no commmutative diagrams in $\mathbb{E}[x_E]$ involving x) and that $D(1) = 1$. Furthermore, the identity on JF is terminal among the other objects in J/JF. Thus the colimit is the sum as claimed. ■

We write $X_{E'}$ for the set $\{\beta : E \longrightarrow E'\}$ when E' is not E, and let $X_E = 1$. Then for any entity F we have $LD(F) = D(F) + X_F$. The unit of the adjunction has as components the injections $D(F) \rightarrowtail LD(F)$.

Proposition 13 *If the algorithm* `INS` *returns* `TRUE`*, then there exists an epi $\phi : LD \longrightarrow D'$ in $\operatorname{Mod}(\mathbb{E}[x_e])$ such that in $\operatorname{Mod}(\mathbb{E})$, with m the evident inclusion,*

$$m : D \rightarrowtail UD' \longleftarrow ULD : U\phi$$

and $(U\phi)\eta_D = m$.

Proof. We will construct ϕ explicitly.

We construct the components $\phi'_E : DE' + X'_E \longrightarrow D'E'$ in two cases.
Case 1: Suppose E' is not equal to E whence $D'E' = DE'$. Let the first component be the identity on DE'. The second component, $X'_E \longrightarrow D'E'$, is defined by $\beta \mapsto D'(\beta)t$.
Case 2: $E' = E$ whence $D'E' = DE + \{t\}$. Let the first component be the inclusion of $DE' = DE$ into $DE + \{t\} = D'E'$. The second component is the constant at t mapping $X'_E \longrightarrow DE + \{t\}$.

As constructed ϕ is a natural transformation. Notice also that each component is epi (in case 2, X'_E is non-empty).

Finally, let $m : D \rightarrowtail UD'$ be given by the evident inclusions and notice $(U\phi)\eta_D = m$. ■

Conversely, consider unrestricted insertions in a database state D — unrestricted in the sense that they may cascade, and result in further inserts. The next proposition says that our algorithm is powerful enough to produce all of these — it just needs to be iterated appropriately. In other words, the algorithm can be used to effect all insertions (as defined in Definition 8).

Proposition 14 *Suppose D'' is a database state, $m : D \rightarrowtail UD''$ a monomorphic natural transformation, and $\phi : LD \longrightarrow D''$ an epimorphic natural transformation, such that $(U\phi)\eta_D = m$, then D'' can be obtained from D by finitely iterated application of the algorithm in Figure 1.*

Proof. Choose an E such that $D''(E)$ is not equal to $D(E)$ but from which all arrows $a : E \longrightarrow E'$, satisfy $D''(E') = D(E')$ Choose also a $t \in D''(E) - D(E)$.

`INS(t, D, E, ins, D')` will ask for the values of $D'(a)t$ and we reply with $U\phi(ULD(a)((U\phi)^{-1}(t)))$ (independent of choice of representative in ϕ^{-1} by naturality of ϕ). Any diagrams that are required to commute will do so since they do in D'', so the `INS` will terminate with `ins = TRUE`.

Now repeat from the choice of E and t above. This terminates by finiteness and acyclicity of the sketch and because the database states are finite set valued, and the resulting $D' = D''$ as required. ■

Now, what about an $\mathbb{E}$ state X such that $m : D \rightarrowtail X \longleftarrow ULD : e$ with $e\eta_D = m$? In fact, all $\mathbb{E}$ states thus 'between' D and ULD can be obtained by iterated insertions.

Corollary 15 *Suppose X is an $\mathbb{E}$ database state, $m : D \rightarrowtail X$ a monomorphism of database states, and $e : ULD \longrightarrow X$ an epimorphism of database states such that $e\eta_D = m$. Then X can be obtained from D by finitely iterated application of the algorithm in Figure 1.*

Proof. X must be of the form UD' since $\mathbb{E}[x_E]$ states are just $\mathbb{E}$ states, together with an extra constant of type E, and $e : ULD \longrightarrow X$ determines that constant as $e(x_E(1))$. Furthermore e must then be of the form $U\phi$ for $\phi : LD \longrightarrow D'$ since, with J being bijective on objects, to give a natural transformation in $\operatorname{Mod}(\mathbb{E})$ is to give a natural transformations in $\operatorname{Mod}(\mathbb{E}[x_E])$, provided only that it is also natural with respect to x_E. Thus we have satisfied the premises of the proposition. ■

5.2 The finite-limit case

We now extend the insert algorithm to deal with finite limit EA sketches. This case covers most real applications since, although the full specification requires limits and coproducts, the latter are usually restricted to attribute domains and these are never updated.

The algorithm (Figure 2) presupposes that the initial insert entity `E` is not the vertex of a limit cone. This is without loss of generality since inserting into the vertex requires inserting into one of the base entities, and (using acyclicity) the appropriate base insert can be done instead. Furthermore, this is the appropriate way to achieve such inserts in practice.

The algorithm is closely related to the previous algorithm (Figure 1). The only significant difference arises when `E` is "minimal" in a cone base. *Minimal* means simply that there is no arrow in the cone base with `E` as its codomain. Inserting into non-minimal cone base entities will not alter the limit, and so can be dealt with by the previous insert algorithm. Inserting into minimal cone base entities results in recursive calls to `INS`. These calls may insert in the cone vertex since the relevant base instances are already being

```
/* INSERT ALGORITHM FOR FINITE LIMIT EA SKETCHES

Algorithm INS(t, D, E, ins, D')
/*  pre: D a database state, E an object of D, t not in D(E).
/* post: if ins is FALSE, D is unchanged, and D' is garbage;
/*       if ins is TRUE t is in D'(E), and D' is a database state for
/*       the extended sketch E[x].

D' = D                 /* D' will be the new database state
D'(E) = D'(E) + { t }  /* this is sum
D'(x_E)1 = t
ins = TRUE

/* The new element needs to have its image given for each arrow out of E as usual.
/* Then if E is a minimal cone base entity, it will determine
/* new elements for vertex.
/* To get them test for compatibility with every possible other tuple.
/* If compatible define all projection values for new vertex element
/* then call insert to add the element to vertex.

for each a : E --> E'
  /* loop completion means all D'(a)t are defined
  if D'(a)t is defined
     next a
  else
     INTER_ASSIGN(D', a, t, ins)
     /* Let user choose D'(a)t's value.
     /* Does not allow new element creation
  if not ins
     exit   /* on fail we return previous state and exit

for each cone C
  if E = C_h(M) for M minimal in base of cone C
  /* C_v, C_h : I --> G are graph homoms, I a finite graph,
  /* G the graph of the sketch, C_v constant,
  /* p : C_h --> C_v ( a nat transf - the projections)
    V = C_v(I)  /* the cone vertex
    for each x in product over B in I of D'(C_h(B))
       with x_{C_h(B)} = t  where C_h(B) = E
       /* an E'th coordinate of x has value t
      if CHECKCOMP(C, D', x)
        for each B in I
          D'(p_B)x = x_{C_h(B)}
        INS(x, D', V, ins, D')
        if not ins
           exit   /* on fail we return previous state and exit

/* end of INS

Algorithm CHECKCOMP(C, D, x) : boolean
/* this checks compatibility for possible new tuple
CHECKCOMP = TRUE
for each B in I
  for each B' in I
    for each Y in I
      for each path p : B --> Y in I
        for each path p' : B' --> Y in I
          if D'(C_h(p))(x_C_h(B)) <> D'(C_h(p'))(x_C_h(B')
             /* C_h extended to functor
             CHECKCOMP = FALSE
             exit
/* end of CHECKCOMP

/* The procedures INTER_ASSIGN, EXTEND_FUNCTION, and CHCD are shown in Figure 1.
```

Figure 2. An insert update algorithm for finite-limit EA sketches

```
/* DELETE ALGORITHM FOR FINITE LIMIT EA SKETCHES

Algorithm REMOVE(t, D, E)
/* removes t and all definitions D(a)t = t'
for a : E --> E'
   D(a)t = undefined   /* means (t,t') removed from the graph of D(a)
D(E) = D(E) - { t }    /* t might already be gone...
/* end of REMOVE


Algorithm DEL(t, D, E)

for each a : E' --> E, for each t' with D(a)t' = t
   DEL(t', D, E')

for each cone C with vertex E
   scan projections p:E-->E' of C for t' not in D(E') with D(p)t = t'
   if no such p exists       /* can there be more than one such p ?
         output(Choose a projection p : E --> E')
         input( p : E --> E' )    /* factor chosen to delete from
         REMOVE(D(p)t, D, E')
         DEL(D(p)t, D, E')

REMOVE(t, D, E)
/* end of DEL
```

Figure 3. A delete update algorithm for finite-limit EA sketches

inserted. The required inserts are determined by explicitly calculating the limit as a collection of tuples.

The delete algorithm for finite-limit EA sketches (Figure 3) is a recursive cascading delete. This is necessary since deleting from either a cone vertex or a cone base entity requires deletion from the other, and this in turn may recur to arbitrary (finite) depth. The order of `REMOVE` and `DEL` and the sequencing of partial functions "`scan the projections...`" is delicate and important.

6. Conclusions

One of the most interesting issues to arise from the application of categorical universal algebra to database systems is the nature of update algorithms in the presence of detailed categorical structure in the form of limits and colimits. This has not arisen previously because universal algebras are rarely updated (we rarely add or remove an element of a given type from an extant algebra), while database updates have only ever been applied to relatively flat systems (systems with many types, but very few limit types).

This paper provides both a precise definition of insert update and update algorithms for categorical sketches including those with finite limits. It makes explicit mathematically the link between the theory and the algorithms. This answers the question which motivated this paper and it will aid in the development of CASE tools needed for quickly developing interoperating sem-structured databases for design.

Although the application of the theory of the sketch data model would seem to depend on the development of the algorithms for mixed sketches, we can in fact make considerable progress using the algorithms for limit sketches since the vast majority of database systems can be designed so that the only colimits are non-updatable (being in fact attribute types).

Nevertheless, the authors have developed algorithms that deal with coproduct sketches, and they are investigating the possibilities for mixed sketches.

7. Acknowledgements

This research was partially supported by the Australian Research Council, the Canadian NSERC, and the Oxford Computing Laboratory. We are grateful to all these sponsors.

References

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

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

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

[4] Aurelio Carboni, Steven Lack, and R. F. C. Walters. Introduction to extensive and distributive categories. *Journal of Pure and Applied Algebra*, 84:145–158, 1993.

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

[6] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13:377–387, 1970.

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

[8] C. N. G. Dampney and Michael Johnson. Fibrations and the DoH Data Model. Unpublished 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 13th International Conference on Software and Systems Engineering and their Applications, Paris, vol 3, 12-5, pp 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. Unpublished report to Caltex, 1993.

[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. J. Date. *Introduction to Database Systems*. Addison-Wesley, sixth edition, 1995.

[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*, to appear, 2001.

[16] Michael Johnson. The Tos, and the Logic of Queries. presented at CATS'94, UTS, 1994.

[17] Michael Johnson. Categorical Logic and Information System Specification. Manuscript, 2000. See also *Journal of the IGPL*, 4, OUP, 1996.

[18] Michael Johnson and C. N. G. Dampney. On the value of commutative diagrams in information modelling. In *Springer Workshops in Computing*, Springer-Verlag, 47–60, 1994.

[19] Michael Johnson and C. N. G. Dampney. Half-duplex interoperations for cooperating information systems. Accepted for CE2001, the Eighth International Conference on Concurrent Engineering, 2001.

[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, 161–166, 2000.

[21] Michael Johnson, Robert Rosebrugh and C.N.G. Dampney. View updates in a semantic data modelling paradigm. *Database Technologies.* Proceedings ADC2001, 29–36, IEEE Computer Society, 2001.

[22] Michael Johnson and Robert Rosebrugh. View updatability based on the models of a formal specification. In press for the Proceedings of Formal Methods Europe (FME01), *Springer Lecture Notes in Computer Science* 2021, 534–549 2001.

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

[24] F.W. Lawvere. *Functorial semantics of algebraic theories*. PhD thesis, Columbia University, 1963.

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

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

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

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