

A Formal Model for Databases with Applications to Schema Merging

Anthony S. Kosky

Department of Computer and Information Sciences

University of Pennsylvania

Philadelphia, PA 19104-6389

kosky@saul.cis.upenn.edu

Abstract

In this paper we present a simple and general model for database schemas and their instances. The model is sufficiently expressive to represent complex, higher order data structures and incorporates representations for specialisation relations and object identity. It is general enough to encode data structures arising from many other semantic data models in a natural way, though we do not attempt to model some of the more sophisticated constraints that occur in other models.

We claim that using a formal mathematical model can help us to understand and deal with various problems arising from database systems, and, to demonstrate this, we present some work on the problem of schema merging that has been carried out using our model.

1 Introduction

In this paper we will develop a new, formal model for database structures and use it to investigate the problem of schema merging. In particular we will concentrate on providing a general model for database schemas which will be similar to the functional data model ([1]), and will support specialisation relationships. Wherever possible we try to minimise the number of different concepts involved in modelling both schemas and database instances, in order to get as simple and uniform a model as possible. We construct a representation of *database instances* which supports object identity, and define what it means for an instance to *satisfy* a database schema. Despite its simplicity, our model is very general and expressive so that database schemas and instances arising from a number of other data models can be translated into it.

We will go on to look at *schema merging*: a problem about which much has been written (see [2]) though the author believes that the formal semantics of such merging processes has not been properly explored or understood. The problem of merging database schemas arises when two or more databases, originating from different sources and possibly containing overlapping information, are combined to form a federated database system (see, for example, [11], [12]), and also in the initial design of a database system, when forming a global database schema from a number of user views of the database. The problem is to find a common view of a number of distinct databases, that is a database schema that, in some sense, *supersedes* the schemas of the databases being merged.

The use of a simple and flexible formal model, such as the one established here, gives us a new insight into the problems of schema merging: we attempt to define an ordering on schemas representing their informational content and define the merge of a collection of schemas to be the least schema which contains all the information of the schemas being merged. However we establish that one cannot, in general, find a meaningful binary merging operator which is associative, though we would clearly require this of any such operator. We rectify this situation by relaxing our definition of schemas, defining a class of *weak schemas* over which we can construct a satisfactory concept of merges. Further we define a method of constructing a *canonical* proper schema with the same informational content as a weak schema whenever possible, thus giving us an adequate definition of the merge of a collection of proper schemas whenever such a merge exists. In addition we show that, if the schemas we are merging are translations from some other data model, our merging process “respects” the original data model. Due to limited space we will only summarise these results here: a more detailed coverage is given in [3].

2 A Model for Database Structures

In this section we will describe a model for databases with complex data structures and object identity. The model will provide a means for representing specialisation, or “is-a” relationships between data structures. It will not attempt to take account of such things as generalisation relationships (see [4]) or various kinds of dependencies, partly due to lack of space and partly because we wish to keep our model sufficiently simple in order for the following sections to be as comprehensible as possible. However the author does not believe that adding such extensions to the model or extending the following theory to take account of them should be overly difficult.

We will describe the model’s features in terms of ER-structures ([5]) since they provide a well known basis for explanations and examples, though the model can be used equally well to represent other semantic models for databases (for a

survey of such models see [1]). In addition to allowing higher order relations (that is relations amongst relations) the model can represent circular definitions of entities and relations, a phenomenon that occurs in some object oriented database systems. Consequently, despite its apparent simplicity, the model is, in some sense, more general and expressive than most semantic models for databases.

2.1 Database Schemas

Digraphs

We will make use of *directed graphs* (*digraphs*) in our representations of both database schemas and instances. We must first give a definition of digraphs which is tailored to the needs of this paper, and consequently may differ a little from definitions that the reader has come across elsewhere. In particular our digraphs will have both their edges and their vertices labelled with labels from two disjoint, countable sets.

Suppose \mathcal{V} and \mathcal{L} are disjoint, countable sets, which we will call the set of *vertex labels* and the set of *arrow labels* respectively. A **digraph** over \mathcal{V} and \mathcal{L} is a pair of sets, $G = (V, E)$, such that

$$\begin{aligned} V &\subseteq \mathcal{V} \\ E &\subseteq V \times \mathcal{L} \times V \end{aligned}$$

If $G = (V, E)$ is a digraph and $p, q \in V$, $a \in \mathcal{L}$ are such that $(p, a, q) \in E$ then we write $p \xrightarrow{a}_G q$, and we may omit the subscript G where the relevant digraph is clear from context.

We will represent database schemas by triples of sets, the first two sets forming a digraph, and the third set representing specialisation relationships between data structures. Before giving a formal definition of database schemas we will explain how we represent a system of data structures (ignoring specialisations) by a digraph.

Representing data structures

Suppose we have a finite set, \mathcal{N} , of **class names** and a finite set, \mathcal{L} , of **attributes** or **arrow labels**. Then we can represent a system of data structures by a digraph $(\mathcal{C}, \mathcal{E})$ over \mathcal{N} and \mathcal{L} satisfying the following condition:

$$\mathbf{A1} \quad p \xrightarrow{a} q \wedge p \xrightarrow{a} r \text{ implies } r \equiv q$$

That is, for any class name $p \in \mathcal{C}$ and any arrow label $a \in \mathcal{L}$, there is at most one class $q \in \mathcal{C}$ such that $p \xrightarrow{a} q$.

If $p, q \in \mathcal{C}$ and $a \in \mathcal{L}$ are such that $p \xrightarrow{a} q$ (that is, $(p, a, q) \in \mathcal{E}$) then we say that p has an a -arrow of class q .

The intuition here, in terms of ER-structures, is that classes, which are the vertices of the digraph, correspond to entity sets, relations and base types, and arrows correspond to the attributes of entities and the *roles* of entities in relations. The concepts of relations/entities/base types and of attributes/roles are therefore unified into two concepts: classes and arrow labels. For example the ER-diagram shown in Figure 1 would be represented by the digraph in Figure 2.

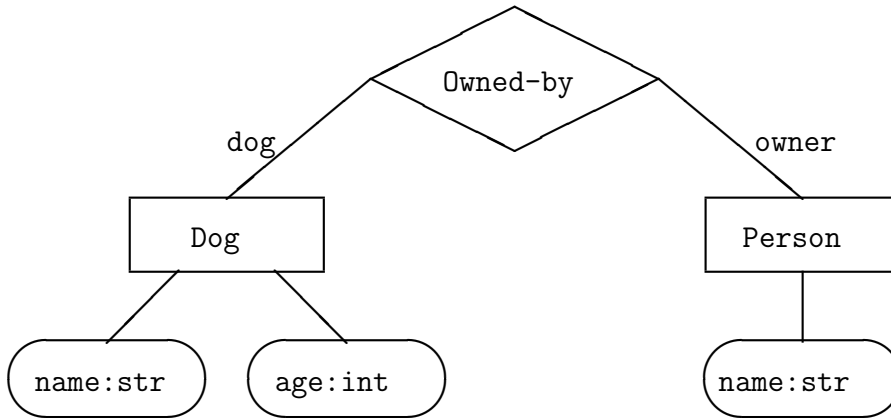


Figure 1: An ER-Diagram

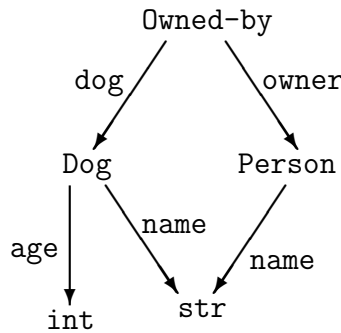


Figure 2: A database schema

Database schemas with specialisation

With the explanation of the use of digraphs given above in mind, we should now be in a position to make sense of our formal definition of database schemas.

Suppose we have a finite set of class names, \mathcal{N} , and a finite set of arrow labels, \mathcal{L} . Then a **database schema** over \mathcal{N} and \mathcal{L} is a triple of sets, $\mathcal{G} = (\mathcal{C}, \mathcal{E}, \mathcal{S})$, where $(\mathcal{C}, \mathcal{E})$ forms a digraph over \mathcal{N} and \mathcal{L} satisfying the axiom **A1** above, and \mathcal{S} is a

partial ordering on \mathcal{C} (that is, it is transitive, reflexive and antisymmetric), such that \mathcal{S} satisfies:

$$\mathbf{A2} \quad \forall p, q, r \in \mathcal{C} \cdot \forall a \in \mathcal{A} \cdot (p, q) \in \mathcal{S} \wedge q \xrightarrow{a} r \text{ implies} \\ (\exists s \in \mathcal{C} \cdot p \xrightarrow{a} s \wedge (s, r) \in \mathcal{S})$$

That is, for every pair $(p, q) \in \mathcal{S}$, if q has an a -arrow of class r , then p also has an a -arrow with some class s where $(s, r) \in \mathcal{S}$. (Note that we write $p \xrightarrow{a}_{\mathcal{G}} q$ to mean $(p, a, q) \in \mathcal{E}$ and we omit the subscript \mathcal{G} when it is obvious from the context).

If $(p, q) \in \mathcal{S}$ then we write $p \implies q$, and say p is a **specialisation** of q . Intuitively what we mean here is that every instance of the class p can also be considered to be an instance of the class q (possibly extended with some additional information). So our axiom, given above, makes sense if we consider it to mean that, in order for us to be able to consider an instance of p to be an instance of q , p must at least have all the arrows of q , and these arrows must have classes for which every instance can be considered to be an instance of class of the corresponding arrow of q .

The conditions **A1** and **A2** are equivalent to those given in [6] and also in [7] for functional schemas (though the former incorporated specialisation relations between arrows, while the later used unlabelled arrows).

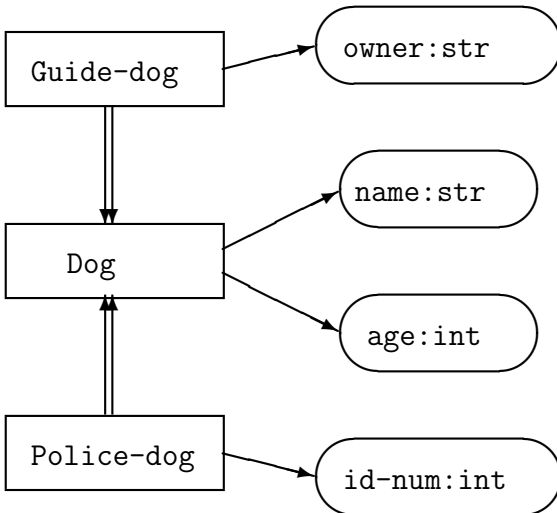


Figure 3: An ER-diagram with “isa” relations

For example the ER-diagram shown in figure 3, where specialisation relationships are marked by double arrows, corresponds to the database schema shown in figure 4, where single arrows are used to indicate edges in \mathcal{E} and double arrows are used to represent pairs in \mathcal{S} .

By putting suitable restrictions on the schemas of our model, we can use them to interpret the schemas of a variety of other data models: relational, entity-

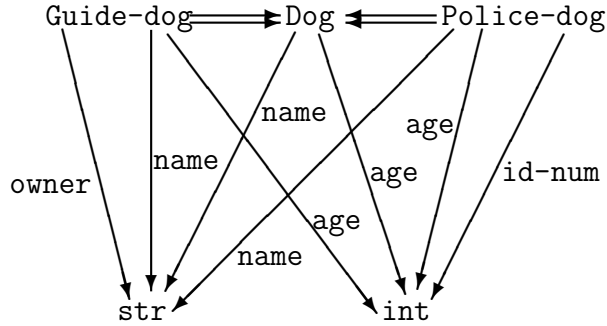


Figure 4: A database schema with “isa” relations

relationship, functional, object-oriented, and so on. For example, in order to interpret a relational database schema, we can stratify \mathcal{C} into two sets of classes, \mathcal{C}_R and \mathcal{C}_A , corresponding to relations and attribute domains respectively, disallow specialization edges, and restrict arrows so that they may only go from classes in \mathcal{C}_R to classes in \mathcal{C}_A . Similarly, in order to interpret ER-schemas, we stratify \mathcal{C} into three sets (attribute domains, entity sets and relationships), and again place certain restrictions on arrows and specialization relations. Such restrictions on schemas can be thought of as *meta-schemas* which the schemas must satisfy. In [3] we define a representation for such meta-schemas and use them to show that the merging process described in Section 3 respects such restrictions.

In an unconstrained form, our schemas are similar to those of the *functional data model* ([6, 8]), though they are not sophisticated enough to interpret data models such as those proposed in [1] or [9], which incorporate constructors for variants. As we develop our data model, we will incorporate the idea of *object identity*, thus making it suitable for modeling the *object oriented databases* described in [10].

Extending schemas with cardinality constraints

If one of our schemas asserts that a class p has an a -arrow of class q , we interpret this as meaning that, for any instance of the schema, if an object of class p has an a -arrow going to some other object, then that object will have class q . However the schema does not tell us whether all objects of class p have a -arrows, or *how many* a -arrows they may have. To include this kind of information, which is common in many database models, we need to assign *cardinality constraints* to arrows.

We will limit ourselves to considering only four different possible **cardinality constraints**, 0-m, 1-m, 0-1 and 1-1, meaning any number of arrows, at least one arrow, at most one arrow, and exactly one arrow respectively. For example the set-valued functions of the model of [1] could be considered to be arrows with the cardinality constraint 0-m, while the normal functions in that model would be

equivalent to arrows with the cardinality constraint 1-1, and a function which could take on null values would be equivalent to an arrow with the cardinality constraint 0-1. It would be possible to have a more complicated and specific set of cardinality constraints, but there is little practical value and no theoretical interest in doing so.

We assume an ordering, \leq , on cardinality constraints, forming the lattice shown in figure 5. So \leq represents the idea of one cardinality constraint being more specific than or implying another.

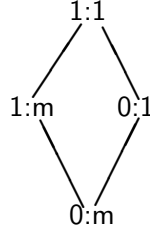


Figure 5: The lattice of cardinality constraints

We extend our schemas with a mapping \mathcal{K} assigning cardinality constraints to arrows. That is

$$\mathcal{K} : \mathcal{E} \rightarrow \{0\text{-}m, 1\text{-}m, 0\text{-}1, 1\text{-}1\}$$

and add the additional axiom

$$\mathbf{A3} \quad \forall p, q, r, s \in \mathcal{C} \cdot \forall a \in \mathcal{L} \cdot p \xrightarrow{a} q \wedge r \xrightarrow{a} s \wedge r \implies p \wedge s \implies q \\ \text{implies } \mathcal{K}(p \xrightarrow{a} q) \leq \mathcal{K}(r \xrightarrow{a} s)$$

This means that, if r is a specialisation of p and p has an a -arrow of class q , then the corresponding a -arrow of r has a cardinality constraint at least as specific as the one from p to q .

2.2 A Model for Database Instances

In this section we will describe a representation for *instances* of databases, and define what it means for an instance to *satisfy* a database schema.

An **instance**, \mathcal{M} , over class names \mathcal{N} and arrow labels \mathcal{L} , is a triple, $(\mathcal{O}, \mathcal{R}, \mathcal{I})$, where \mathcal{O} is a countable set of **object identities**, $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{L} \times \mathcal{O}$ is such that $(\mathcal{O}, \mathcal{R})$ forms a digraph over \mathcal{O} and \mathcal{L} , and $\mathcal{I} : \mathcal{O} \rightarrow \mathcal{N}$ (that is \mathcal{I} maps object identities to class names) is said to be the **interpretation** of the object identifiers. (Once again we will use notation $o \xrightarrow{a}_{\mathcal{M}} o'$ to mean $(o, a, o') \in \mathcal{R}$, where $\mathcal{M} = (\mathcal{O}, \mathcal{R}, \mathcal{I})$).

The idea here is that object identifiers correspond to the real world objects represented in the database, and the relation \mathcal{R} represents how these objects are related by the attributes in \mathcal{L} . The interpretation, \mathcal{I} , shows the least class (under the

ordering \mathcal{S}) to which a particular object belongs. An object can also be considered as belonging to any superclasses of this class, so that, for the schema shown in Figure 4, every object belonging to the class `police_dog` also belongs to the class `dog`.

A database model, $\mathcal{M} = (\mathcal{O}, \mathcal{R}, \mathcal{I})$, is said to **satisfy** a database schema $\mathcal{G} = (\mathcal{C}, \mathcal{E}, \mathcal{S}, \mathcal{K})$ iff

- S1** $\forall o, o' \in \mathcal{O} \cdot \forall p, q \in \mathcal{C} \cdot \forall a \in \mathcal{L} \cdot$
 $p = \mathcal{I}(o) \wedge p \xrightarrow{a}_{\mathcal{G}} q \wedge o \xrightarrow{a}_{\mathcal{M}} o' \text{ implies } \mathcal{I}(o') \implies q$
- S2** $\forall o \in \mathcal{O} \cdot \forall p, q \in \mathcal{C} \cdot \forall a \in \mathcal{L} \cdot p = \mathcal{I}(o) \wedge p \xrightarrow{a}_{\mathcal{G}} q \wedge 1\text{-}m \leq \mathcal{K}(p \xrightarrow{a} q)$
 $\text{implies } (\exists o' \in \mathcal{O} \cdot o \xrightarrow{a}_{\mathcal{M}} o')$
- S3** $\forall o, o', o'' \in \mathcal{O} \cdot \forall p, q \in \mathcal{C} \cdot \forall a \in \mathcal{L} \cdot$
 $p = \mathcal{I}(o) \wedge p \xrightarrow{a}_{\mathcal{G}} q \wedge o \xrightarrow{a}_{\mathcal{M}} o' \wedge o \xrightarrow{a}_{\mathcal{M}} o'' \wedge 0\text{-}1 \leq \mathcal{K}(p \xrightarrow{a} q)$
 $\text{implies } o' = o''$

The first condition says that, if a schema specifies an a -arrow of some class p , and an object of class p has an a -arrow then the object connected to it by the a -arrow must belong to the class specified by the schema. For example, again for the schema in Figure 4, if a dog has a name and an age recorded in the database, then these must belong to the classes `str` and `int` respectively.

The second and third conditions mean that, if class p has an a -arrow of class q , then if the arrow has associated cardinality constraint 1- m or 1-1, then for every object of class p must have an a -arrow going to an object of class q , while, if the arrow has a cardinality constraint of 0-1 or 1-1, then any object of class p can have at most one a -arrow.

A simple example

In order to clarify our idea of models we will give a brief example. We will start with the schema shown in figure 6 (certain implicit edges, namely those induced on the class `Police_dog` by its specialisation relationship to the class `Dog` have been omitted). In this diagram cardinality constraints have been marked in small type at the ends of the arrows.

We consider the following set of object identities:

$$\mathcal{O} = \{\text{Fido, Bonzo, Rover, Humphrey, George, Poodle, German-Shepherd, Old-English-Sheepdog, Rover-id}\}$$

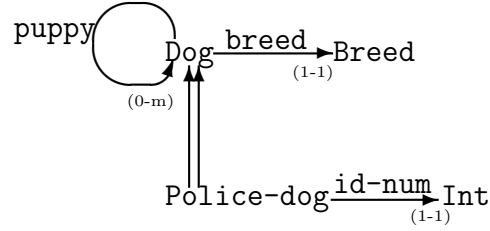


Figure 6: A simple example

The classes of these object identities are given by:

$$\begin{aligned}
 \mathcal{I}(\text{Fido}) &= \mathcal{I}(\text{Bonzo}) = \mathcal{I}(\text{Humphrey}) = \mathcal{I}(\text{George}) = \text{Dog} \\
 \mathcal{I}(\text{Rover}) &= \text{Police-dog} \\
 \mathcal{I}(\text{Poodle}) &= \mathcal{I}(\text{German-Shepherd}) = \mathcal{I}(\text{Old-English-Sheepdog}) = \text{Breed} \\
 \mathcal{I}(\text{Rover-id}) &= \text{Int}
 \end{aligned}$$

and our attribute relations are

$$\begin{aligned}
 \mathcal{R} = \{ & \text{Fido} \xrightarrow{\text{breed}} \text{Poodle}, \text{Bonzo} \xrightarrow{\text{breed}} \text{Poodle}, \text{Rover} \xrightarrow{\text{breed}} \text{German-Shepherd}, \\
 & \text{Humphrey} \xrightarrow{\text{breed}} \text{Old-English-Sheepdog}, \text{George} \xrightarrow{\text{breed}} \text{Poodle}, \\
 & \text{Fido} \xrightarrow{\text{puppy}} \text{Bonzo}, \text{Fido} \xrightarrow{\text{puppy}} \text{George}, \text{Rover} \xrightarrow{\text{id-num}} \text{Rover-id}, \}
 \end{aligned}$$

Note that the cardinality constraints are satisfied: each dog has exactly one breed, and each police dog has exactly one id number.

3 An Application to Schema Merging

Having developed a flexible, formal model for database systems, we can now go on to use it to interpret and express various phenomenon which occur when studying or working with databases. Giving a mathematical formulation of such phenomenon helps us to clarify the problem and, in addition, can help bring to light various complexities and problems which might remain unnoticed if we studied the phenomenon in an informal way. Two such applications of the model were studied in detail in [3], and considerable insights into the problems were gained. First the model was extended by associating *printable values* to objects, and a theory of the *observable behaviour* of databases was developed and discussed. Also the problem of merging database schemas was investigated and a mathematically precise definition of the merge of a set of database schemas was constructed. In this section we will summarise the results on schema merging given in that paper.

3.1 Merging Database Schemas

Much has been written on the subject of merging database schemas (see [2] for a survey) and a number of tools and methodologies have been developed. These range from sets of tools for manipulating schemas into some form of consistency, [7, 11], to algorithms which take two schemas, together with some constraints, and produce a merged schema [13]. However, to the best of our knowledge, the question of what meaning or semantics this merging process should have has not been adequately explored. Indeed, several of the techniques that have been developed are *heuristics*: there is no independent characterisation of what the merge should be.

The first problem to be resolved in forming such a common schema is deciding on the correspondences between the classes of the various databases. This problem is inherently *ad hoc* in nature, and is dependent on the real-world interpretations of the databases, so that such information must be provided by the designer of the system.

In the study of merging in [3], the only correspondences between classes considered are equivalence, where classes from two separate database schemas represent the same class of real world objects, and subclass correspondences, where a class from one schema is included in a class from another. The interpretation that the merging process places on names is that if two classes in different schemas have the same name, then they are the same class, regardless of the fact that they may have different arrow edges. Subclass correspondences are represented by adding an extra *correspondence* schema to the set of schemas being merged which encodes such correspondences as specialisations between the classes. The designer of the system is, therefore, called upon to resolve naming conflicts, whether homonyms or synonyms, by renaming classes and arrows where appropriate. For example, if one schema has a class *Dog* with arrow edges *License#*, *Owner* and *Breed*, and another schema has a class *Dog* with arrow edges *Name*, *Age* and *Breed*, then the merging process will collapse them into one class with name *Dog* and arrow edges *License#*, *Owner*, *Name*, *Age*, and *Breed*, with the potential result that null values are introduced¹. However, if the user intends them to be separate classes, the classes must be renamed (for example, *Dog1* and *Dog2*), and, if the user intends *Dog1* to be a subclass of *Dog2* then he/she should include the classes *Dog1* and *Dog2* and a specialisation relation between *Dog1* and *Dog2* in the correspondence schema.

What we want to find is a schema that presents all the information of the schemas being merged, but no additional information. Hence we want the “*least upper bound*” of the database schemas under some sort of information ordering. Recall that, in addition to defining a view of a database, a database schema expresses

¹This contrasts with [11], which states that entities *should* be renamed, and specialization edges introduced to *avoid* introducing null values

certain requirements on the structure of the information stored in the database. When we say that one database schema presents more information than another, we mean that any instance of the first schema could be considered to be an instance of the second one. The first schema must, therefore, dictate that any database instances must contain at least all the information necessary in order to be an instance of the second schema.

Our approach is to first find a way to describe the merge of exactly two database schemas. This can be thought of as providing a binary operator on database schemas, which, provided it is associative and commutative, can then be extended to a function on finite sets of database schemas in the obvious manner. Such a binary merge is defined as the join (least upper bound) of two schemas under some suitable information ordering.

Some problems with finding merges of schemas

Having developed some intuitive idea of what the merge of two databases should be, we must then go on to find some formal definition of an information ordering on schemas and hence of merges of schemas. We proceed via a series of experiments and examples, in order to try to get a better idea of what the ordering should be. However it soon becomes apparent that things are more complicated than we might have hoped at first.

One of the first problems we notice is that the merge of two schemas may contain extra *implicit* classes in addition to the classes of the schemas being merged. For example figure 7 shows two schemas being merged. The first schema asserts that the class `Guide-dog` is a subclass of both the classes `Dog` and `Working-animal`. The second schema asserts that the classes `Dog` and `Working-Animal` both have `name`-arrows of classes `Dog-Name` and `Animal-Name` respectively. Combining this information, as we must when forming the merge, we conclude that `Guide-Dog` must also have a `name`-arrow and that this arrow must be of both the class `Dog-name` and `Animal-Name`. Consequently, due to the restrictions in our definition of database schemas in Section 2.1, the `name`-arrow of the class `Guide-Dog` must have a class which is a specialisation of both `Dog-Name` and `Animal-Name` and so we must introduce such a class into our merged schema.

When we consider these implicit classes further we find that it not sufficient merely to introduce extra classes into a schema with arbitrary names: the implicit classes must in fact be treated differently from normal classes. Firstly, if we were to give them the same status as ordinary classes we would find that binary merges are not associative. This is because, having introduced a implicit class into the merge of two schemas, merging with a third schema may cause additional implicit classes to be introduced which are specialisations of the first class. Consequently the order in which schemas are chosen for merging affects the order in which implicit classes

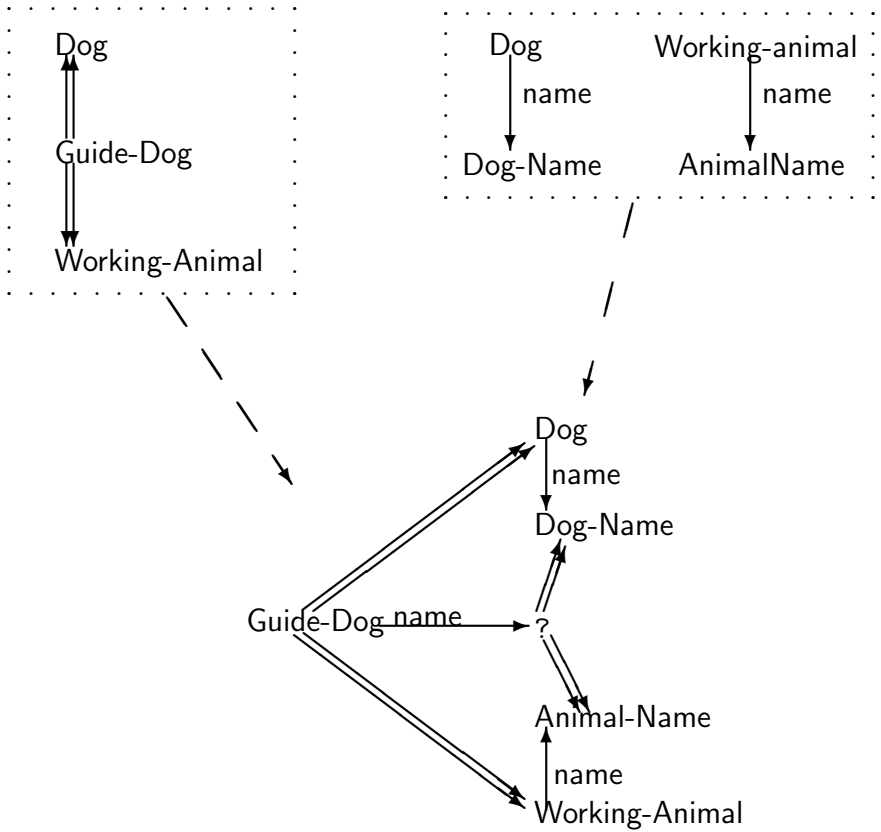


Figure 7: Schema merging involving implicit classes

are introduced and how they are related to one another.

Another problem is that, in our as yet hypothetical information ordering on schemas, it is possible for one schema to present more information than another without containing as many implicit classes. It is clear that for one schema to

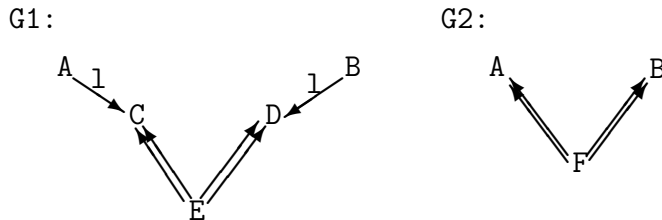
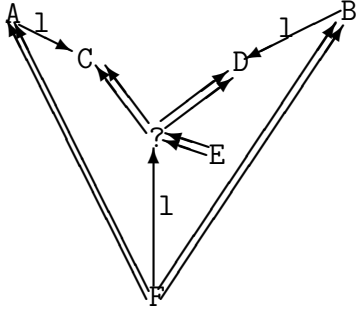


Figure 8: Yet more schemas

present all the information of another (plus additional information) it must have, at least, all the normal classes of the other. However let us consider the two schemas shown in figure 8. We would like to assert that the schema **G3** shown in figure 9 is the merge of the two schemas, **G1** and **G2**. However the schema **G4** also presents all the information of **G1** and **G2**, and in addition contains fewer classes than **G3**. The point is that **G4** asserts that the 1-arrow of **F** has class **E**, which may impose

G3:



G4:

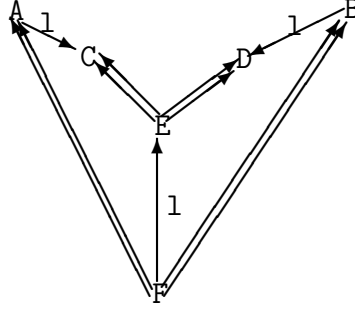


Figure 9: Possible candidates for the merges of the schemas

restrictions on it in addition to those which state that it’s a subclass of both C and D, while G3 only states that the 1-arrow of F has both classes C and D.

We could extend our definition of database schemas to allow special implicit classes, and then try to find an ordering which takes account of them. However such a treatment would be very complicated, and so we adopt a slightly different approach.

3.2 Weak Schemas

In order to avoid the complexities of introducing implicit classes, what we do is first weaken our definition of database schemas so that these classes become unnecessary (they are indeed *implicit*). We then define an information ordering on these *weak schemas*, such that binary joins exist and are associative; and finally we present a method of converting a weak schema into a proper schema by introducing additional “implicit” classes. The idea is that we can do all our merging using weak schemas, and then convert the results to proper schemas when we are done.

We define a **weak schema** over class names \mathcal{N} and arrow labels \mathcal{L} in a similar manner to our definition of (*proper*) schemas in Section 2.1, except that we relax the axiom **A1** in the following manner: instead of requiring that \mathcal{E} be functional in its first two arguments, we adopt the weaker requirement that for any class p and arrow label a , and any two distinct classes q and r , if $p \xrightarrow{a} q$ and $p \xrightarrow{a} r$ then q and r may not be specialisations of one-another. Hence, for any $p \in \mathcal{C}$ and any $a \in \mathcal{L}$, the set $\{q \mid p \xrightarrow{a} q\}$ is a *co-chain* under the ordering \mathcal{S} .

Note that any proper database schema, according to the definitions in section 2.1, is also a weak schema according to this definition.

We define an ordering \sqsubseteq on weak schemas, such that $\mathcal{G}_1 \sqsubseteq \mathcal{G}_2$ iff all the classes of \mathcal{G}_1 are also classes of \mathcal{G}_2 , each specialisation edge in \mathcal{G}_1 is also in \mathcal{G}_2 , and for each class p in \mathcal{G}_1 , if p has an a -arrow of class q in \mathcal{G}_1 , then p has a corresponding a -arrow in \mathcal{G}_2 which is at least as specific: that is, which has at least as specific a cardinality

constraint and which has a class which is a specialisation of q in \mathcal{G}_2 .

It can be shown that \sqsubseteq is a partial ordering on weak schemas, and, further, that if $\mathcal{G}_1 \sqsubseteq \mathcal{G}_2$, then every instance of \mathcal{G}_1 can also be considered to be an instance of \mathcal{G}_2 via a natural projection.

In [3] it is proven that the ordering \sqsubseteq on weak schemas is bounded complete. That is, for any weak schemas \mathcal{G}_1 and \mathcal{G}_2 , if there exists a weak schema \mathcal{G}' such that $\mathcal{G}_1 \sqsubseteq \mathcal{G}'$ and $\mathcal{G}_2 \sqsubseteq \mathcal{G}'$ then there is a least such weak schema $\mathcal{G}_1 \sqcup \mathcal{G}_2$. Further the proof is constructive so that it can be used as an algorithm to construct $\mathcal{G}_1 \sqcup \mathcal{G}_2$ from \mathcal{G}_1 and \mathcal{G}_2 . Since \sqsubseteq is a partial order, the operator \sqcup is associative, commutative and idempotent, and so can be repeatedly applied in order to find the least upper bound of any collection of weak schemas whenever it exists.

It is shown that the least upper bound of a collection of weak schemas exists whenever their specialisation relations satisfy some *compatibility constraints*, and such a collection of weak schemas is said to be **compatible**. Consequently we have a way of formulating a weak schema which contains all the information contained in any compatible collection of schemas but no additional information, and can therefore be considered to be the **weak schema merge** of the collection of schemas. For example the two schemas, G1 and G2, shown in figure 9 are compatible and their

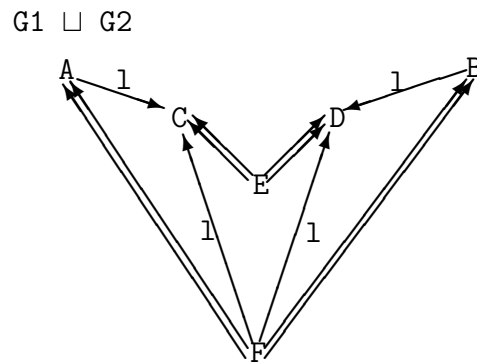


Figure 10: The least upper bound of two schemas

least upper bound is the weak schema shown in figure 10.

Having found the weak schema merge of a collection of schemas, we must then introduce the necessary implicit classes into the weak schema in order to make it into a proper schema according to the definition in section 2.1. An algorithm for doing this is presented in detail in [3]. Basically we start by recursively finding all the sets of classes, X , such that, for some class p and series of arrow labels a_1, \dots, a_n , we have $p \xrightarrow{a_1} \dots \xrightarrow{a_n} q$ for every class q in X . For each such set X we check to see if it has a unique minimal element, and, if not, we introduce an implicit class which is a specialisation of all the classes in X and make this class inherit the arrows of all the

classes in X . Finally we rearrange the arrows of the resulting weak schema so that, if some class u has several a -arrows then we replace them with a single a -arrow to the implicit class which is a specialisation of the classes of all the original a -arrows.

If $\bar{\mathcal{G}}$ is the schema built from \mathcal{G} using this algorithm, then we show that $\bar{\mathcal{G}}$ is indeed a proper schema, that $\mathcal{G} \sqsubseteq \bar{\mathcal{G}}$, and claim that, in some sense, $\bar{\mathcal{G}}$ is the least such proper schema (except for the possible inclusion certain unnecessary but inconsequential specialisation edges). Consequently, given a collection of compatible proper schemas, we define their **merge** to be the proper schema constructed from their weak schema merge in this manner.

Finally, in [3], we define *meta-schemas* to represent the restrictions imposed on the structure of schemas by various data models. A meta-schema divides the class names into a number of disjoint sets, such as entity-sets, relationships and attribute domains, and then imposes restrictions on which arrows may go from classes in each set to classes in each other set. We show that our merging process respects meta-schemas: that is, given a compatible collection of schemas satisfying some meta-schema, their merge will also satisfy the meta-schema. We conclude that we can apply our definition of merging to the schemas of other, established data models by first translating them into our model, then merging them, and then translating the result back into the original data model.

4 Conclusions and Further Work

While the model for database schemas and their instances presented in this paper is both simple and extremely general, there are a number of concepts in the literature for database systems which are potentially useful but which have not been incorporated in this model. These include such things as generalisation relationships, functional dependencies, existence dependencies and so on. The model does incorporate representations for specialisation relations and cardinality constraints, and the author believes that representations for these other concepts would be no more difficult to add. However we have chosen to keep the model relatively simple so as to make it as easy to reason about formally as possible.

By weakening the definition of schemas, we have been able to define the merge of a collection of *weak schemas* as their least upper bound under an information ordering. We can then form a *proper schema* from a weak schema by introducing additional *implicit classes* as necessary, thus giving us a way of defining the merge of a collection of proper schemas. Although the number of implicit classes introduced in the examples we have looked at so far have been small, we have not investigated how many might be introduced in general. It may be possible to construct pathological examples where the number of implicit classes required is huge, though the author believes that such examples are unlikely to arise in practice.

There may well be equally valid, alternative interpretations of what the merge of a collection of schemas should be, though we believe that, in order for some concept of a schema merge to be useful, it should have a similar, simple and intuitive definition in terms of a formal data model such as the one presented here. Consequently we believe that the methods used in this paper should be equally applicable to other such concepts of merging databases. In general we feel that this paper illustrates the potential benefits of using mathematical techniques and uniform, well-defined models in investigating the underlying theory of database systems.

Finally the author has noticed a number of similarities between the various levels of information considered (instances, schemas, meta-schemas, etc.) and their relationships to one another, and would like to investigate the possibility of forming a still more uniform and general model capable of spanning these various information levels.

Acknowledgements: I would like to thank Peter Buneman for his original ideas and advice, and also Susan Davidson for numerous comments, help and advice.

References

- [1] R. Hull and R. King. Semantic database modelling: survey, applications and research issues. *ACM Computing Surveys*, 19(3):201–260, September 1987.
- [2] C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, December 1986.
- [3] A. Kosky. *Modeling and Merging Database Schemas*. Technical Report, University of Pennsylvania, 1991.
- [4] John Miles Smith and Diane C. P. Smith. Database abstractions: aggregation and generalisation. *ACM Transactions on Database Systems*, 2(2):105–133, June 1977.
- [5] Peter Pin-Shan Chen. The entity-relationship model — toward a unified view of data. *ACM Trans. on Database Systems*, 1(1):9–36, March 1976.
- [6] U. Dayal and H. Hwang. View definition and generalisation for database integration in multibase. *IEEE Transactions on Software Engineering*, SE-10(6):628–644, November 1984.
- [7] A. Motro. Superviews: virtual integration of multiple databases. *IEEE Transactions on Software Engineering*, Vol. SE-13(7):785–798, July 1987.

- [8] D. Shipman. The Functional Data Model and the Data Language DAPLEX. *ACM Transactions on Database Systems*, 6(1):140–173, March 1981.
- [9] A. Ohori. Semantics of Types for Database Objects. *Theoretical Computer Science*, 76:53–91, 1990.
- [10] F. Bancilon. Object-oriented database systems. In *Principles of Database Systems*, pages 152–162, 1988.
- [11] J. Smith, P. Bernstein, U. Dayal, N. Goodman, T. Landers, K. Lin, and E. Wong. Multibase— Integrating Heterogeneous Distributed Database Systems. In *Proceedings of AFIPS*, pages 487–499, 1981.
- [12] A. Sheth and J. Larson. Federated database systems for managing distributed heterogeneous and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.
- [13] A. Sheth, J. Larson, J. Cornello, and S. Navethe. A tool for integrating conceptual schemas and user views. In *Proceedings of 4th International Conference on Data Engineering*, pages 176–183, 1988.