September 1991

Modeling and Merging Database Schemas

Anthony S. Kosky
University of Pennsylvania

Modeling and Merging Database Schemas

Abstract
We define a general model for database schemas which is basically functional and supports specialisation relationships. Despite it's 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 the model.

We define and investigate a representation for the observations that can be made by querying a database system, and, in particular, look at which observations are valid for a particular database schema, and when one observation implies the observability of another. We will also look at the correspondence between the instances of a database schema and the observations that can be made for the database.

We then go on to look at the problem of schema merging: we define an ordering on schemas representing their informational content and define the merge of a collection of schemas to be the least schema with the informational content of all 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 can exist. 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.

Comments
Database Research Group.

Modeling and Merging Database Schemas *

Anthony S. Kosky
Department of Computer and Information Sciences
University of Pennsylvania
Philadelphia, PA 19104-6389

25 September 1991

1 Introduction

In this paper we will construct and investigate various aspects of a new, formal model for database structures. In particular we will concentrate on providing a general model for database schemas which will be basically functional ([HK87]), 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 the model.

We will define and investigate a representation for the observations that can be made by querying a database system, and, in particular, look at which observations are *valid* for a particular database schema, and when one observation implies the observability of another. We will also look at the correspondence between the instances of a database schema and the observations that can be made for the database.

We will go on to look at *schema merging*: a problem about which much has been written (see [BLN86]) though the author believes that the formal semantics of such merging processes has not been properly explored or understood. The use of a simple and flexible formal model, such

*This research was supported in part by ARO DAAL03-89-C-0031PRIME, ARO DAAL03-89-C-0031SUB5 and NSF IRI 8610617

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 with the informational content of all 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 can exist. 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.

The paper is organised as follows: in Section 2 we introduce our data model; in Section 3 we construct a model of observations of databases, while in Section 4 we develop a correspondence between these observations and the models of database instances developed in Section 2.2; and in Sections 5 and 6 we look at schema merging. In Section 5.1 we will discuss what properties we would like the merge of a collection of schemas to satisfy and demonstrate some problems with constructing such a merge, while in Section 5.2 we introduce the concept of *weak schemas* and use them to define a satisfactory concept of merges. In Section 5.3 we show how to control the merging process in order to make the merge of some schemas reflect various correspondences between the concepts expressed in the individual schemas, and in Section 5.4 we define the concept of *meta-schemas* and use them to show that our merging process respects various other data-models. In Section 6 we define the concept of *lower merges* corresponding to the intersection of the information contained in a collection of schemas, rather than the sum of information represented be the merges defined in Section 5. We conclude in Section 7.

This paper is written in a modular fashion, so that it should not be necessary for the reader to read those sections concerning aspects of the paper that he is not interested in.

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 [SS77]) 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 extentions 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 ([Che76]) 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 [HK87]). 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:

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

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.

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:

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

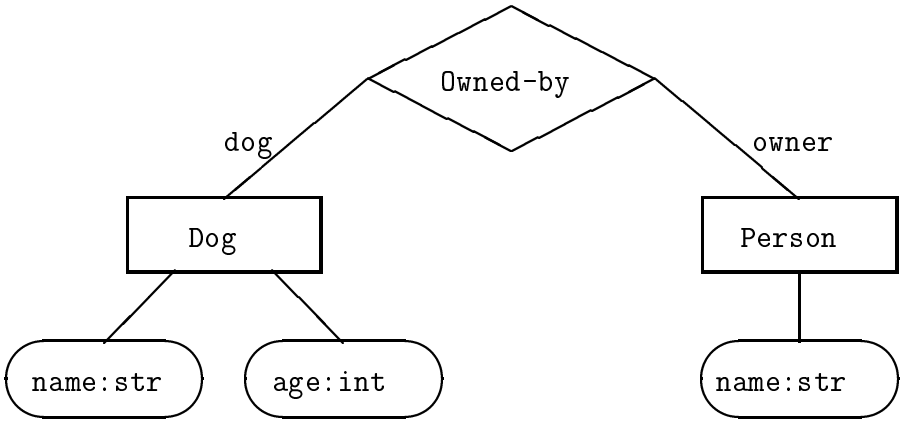


Figure 1: An ER-Diagram

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 $p \Longrightarrow q$, 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 [DH84] and also in [Mot87] for functional schemas (though the former incorporated specialisation relations between arrows, while the later used unlabelled arrows).

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-relationship, functional, object-oriented, and so on. For example, in order to interpret a relational database schema, we

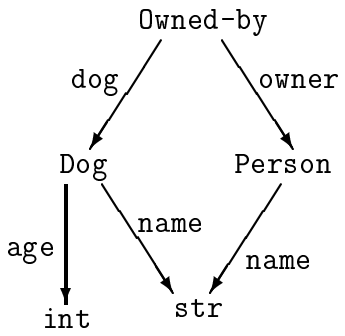


Figure 2: A database schema

can stratify $\mathcal{C}$ into two sets of classes, $\mathcal{C}_R$ and $\mathcal{C}_A$, corresponding to relations and attribute domains respectively, dissallow 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. We will describe this sort of restriction on schemas in detail in section 5.4. The ability to embed other data models in our data model is important since, in section 5, we will show how to merge the schemas of our model, and, using various embeddings, this merging technique can then be applied to other data models.

In an unconstrained form, our schemas are similar to those of the *functional data model* ([DH84, Shi81]), though they are not sophisticated enough to interpret data models such as those proposed in [HK87] or [Oho90], 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 [Ban88].

Extending schemas with arity 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 *arities* to arrows.

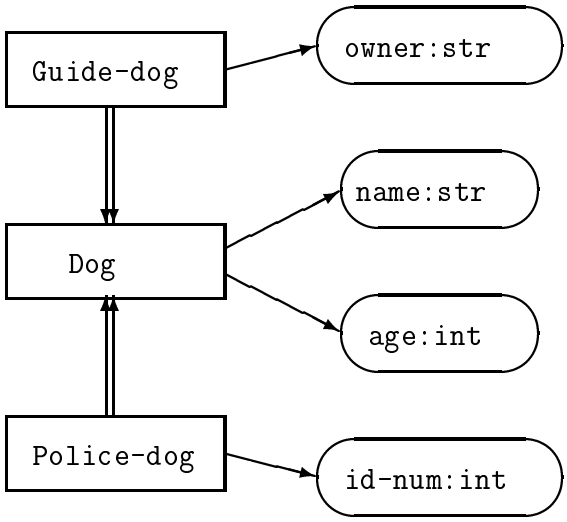


Figure 3: An ER-diagram with "isa" relations

We will limit ourselves to considering only four different possible **arity 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 [HK87] could be considered to be arrows with the arity 0-m, while the normal functions in that model would be equivalent to arrows with the arity 1-1, and functions which could take on null values would be equivalent to arrows with the arity 0-1. It would be possible to have a more complicated and specific set of arity constraints, but there is little practical value and no theoretical interest in doing so.

We assume an ordering, $\leq$ on arity constraints forming the lattice shown in figure 5. So $\leq$ represents the idea of one arity constraint being less specific than or a consequence of another.

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

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

and add the additional axiom

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

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 an arity at least as specific as the one from p to q.

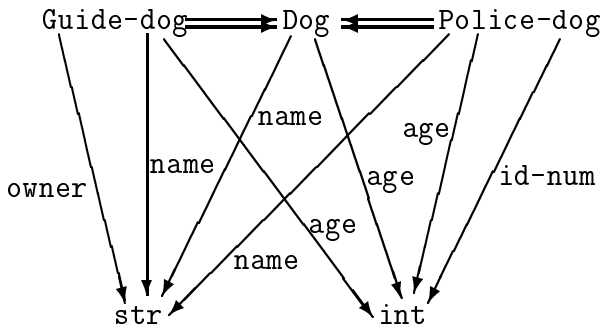


Figure 4: A database schema with "isa" relations

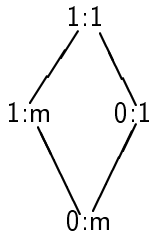


Figure 5: The lattice of arity constraints

Associating printable values with classes

The database schemas described so far provide a structure which we can use to represent the internal structure of data in a database. However they do not provide any means of representing how data items may appear to a user of the database. For this purpose we will associate sets of *printable values* with classes. It should be noted that, in certain data models, not all classes have values associated with them which can be displayed to the user: for example in ER-structures, only the attribute domains have actual printable values associated with them, the other classes (entity sets and relationships) being represented as products of their attributes. In keeping with our philosophy of trying to obtain as general and uniform a model as possible, we will allow sets of printable values to be associated with all classes, on the basis that associating a set containing only one printable value with a class does not allow

us to convey any information using the printable values for that class, and hence is equivalent to not associating any possible representations with the class at all.

We start by assuming a set $\mathcal{D}$ of **printable values**. For each class $p \in \mathcal{C}$ we associate a *non-empty* set $\mathcal{D}^p \subseteq \mathcal{D}$ said to be the **printable value set** of p.

We put the additional restriction on schemas that

$$\textbf{A4}\ \ \forall p, q \in \mathcal{C} \cdot p \Longrightarrow q \ \text{ implies } \ \mathcal{D}^p \subseteq \mathcal{D}^q$$

That is, if p is a specialisation of q, then $\mathcal{D}^p$ is a subset of $\mathcal{D}^q$.

We now extend our database schemas to be five-tuples, $\mathcal{G} = (\mathcal{C}, \mathcal{E}, \mathcal{S}, \mathcal{K}, \mathcal{D}^{\mathcal{C}})$, where $\mathcal{D}^{\mathcal{C}} : \mathcal{C} \to \mathcal{P}(\mathcal{D})$ is the mapping from classes to non-empty subsets of $\mathcal{D}$ given by

$$\mathcal{D}^{\mathcal{C}}(p) \ = \ \mathcal{D}^p$$

For example, for the database schema shown in figure 4, we could take $\mathcal{D}$ to be the set of all alpha-numeric strings. We could then take $\mathcal{D}^{\texttt{str}} = \mathcal{D}$ and $\mathcal{D}^{\texttt{int}}$ to be the set of strings of digits representing integers. For each other class p in the schema, where we wouldn't want any printable information directly associated with items in the class (as against being associated with the attributes of items), we could set $\mathcal{D}^p$ to be the set contaning just the empty string.

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. We will first describe the instances without printable values, in order to explain the basic concept, and will then extend them in order to take account of printable values.

A **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 $\mathcal{I} : \mathcal{O} \to \mathcal{N}$ (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, $\mathcal{I}$, represents the least class of object, though it is also possible to consider an object to belong to any classes of which its interpretation is a specialisation.

For example, in a database for the schema shown in Figure 4, an object with interpretation `police-dog` could be considered to be of class `police-dog` or of class `dog`.

A database model, $\mathcal{M} = (\mathcal{O}, \mathcal{R}, \mathcal{I})$, is said to **satisfy** a database schema $\mathcal{G} = (\mathcal{C}, \mathcal{E}, \mathcal{S}, \mathcal{K})$ (ignoring printable value sets for the time being) iff

S1 $\forall o, o' \in \mathcal{O} \cdot \forall p, q \in \mathcal{C} \cdot \forall a \in \mathcal{L} \cdot$
$\mathcal{I}(o) = p \wedge p \xrightarrow{a}_{\mathcal{G}} q \wedge o \xrightarrow{a}_{\mathcal{M}} o'$ implies $\mathcal{I}(o') \Longrightarrow q$

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

S3 $\forall o, o', o'' \in \mathcal{O} \cdot \forall p, q \in \mathcal{C} \cdot \forall a \in \mathcal{L} \cdot$
$\mathcal{I}(o) = p \wedge p \xrightarrow{a}_{\mathcal{G}} q \wedge o \xrightarrow{a}_{\mathcal{M}} o' \wedge o \xrightarrow{a}_{\mathcal{M}} o'' \wedge \text{0-1} \leq \mathcal{K}(p \xrightarrow{a} q)$
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 have the class specified by the schema. For example, 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 arity 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 arity of 0-1 or 1-1, then any object of class p can have at most one a-arrow.

Modeling printable values

As with our initial study of database schemas (Section 2.1), we have so far only described models for the internal representation of information in a database, without taking into account the *printable values* associated with objects in the database. It is clearly necessary to have some kind of observable representation associated with certain items in a database, since the rest of the model serves only to represent the correspondences and relationships between items of data, but does not determine how this information may be conveyed to the user. However some data models choose to associate observable representations with only some classes of data items: for example, in the relational model, only the attribute domains have observable values associated with them directly. In addition many models choose to equate data items if they, and their attributes, have the same observable representations (this is true of the relational, entity-relationship and functional models). Once again we will opt

for a more general and uniform approach which can be used to interpret these models via the enforcement of various constraints. We will allow (and in fact require) *printable values* to be associated with all items in the database, and will adopt the concept of *object identity* for our models, allowing a number of objects with similar printable values to be stored in a database simultaneously.

Once again we assume a set $\mathcal{D}$ of *printable values*. We extend our models to be four-tuples of the form $\mathcal{M} = (\mathcal{O}, \mathcal{R}, \mathcal{I}, \mathcal{V})$, where $(\mathcal{O}, \mathcal{R}, \mathcal{I})$ is a model as described earlier, and $\mathcal{V} : \mathcal{O} \to \mathcal{D}$ is a function from object identities to printable values. For any object identity, $o \in \mathcal{O}$, $\mathcal{V}(o)$ is said to be the **value** of o in $\mathcal{M}$.

We must also extend our definition of what it means for a model to satisfy a schema. A model, $\mathcal{M} = (\mathcal{O}, \mathcal{R}, \mathcal{I}, \mathcal{V})$, is said to **satisfy** a schema (with printable value sets), $\mathcal{G} = (\mathcal{C}, \mathcal{E}, \mathcal{S}, \mathcal{K}, \mathcal{D}^{\mathcal{C}})$, iff, in addition to conditions **S1** to **S3** stated earlier, the following condition holds:

S4 $\forall o \in \mathcal{O} \cdot \forall p \in \mathcal{C} \cdot (\mathcal{I}(o) = p)$ implies $(\mathcal{V}(o) \in \mathcal{D}^p)$

That is, for any object identity o, of class p, the value of o is in the printable value set of p.

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 `Bear` by its specialisation relationship to the class `Animal` have been omitted). In this diagram arities have been marked in small type at the ends of the arrows. We will take $\mathcal{D}$, our printable values, to be the set of alpha-numeric strings, and to make things easy we will set $\mathcal{D}^p = \mathcal{D}$ for each class p in the schema.

We consider the following set of object identities:

$$\begin{aligned}\mathcal{O} \;=\; \{&\mathsf{Pooh},\ \mathsf{Piglet},\ \mathsf{Eeyore},\\ &\mathsf{Pooh\text{-}house},\ \mathsf{Piglet\text{-}house},\ \mathsf{Eeyore\text{-}house},\\ &\mathsf{hunny},\ \mathsf{condensed\text{-}milk},\ \mathsf{haycorns},\ \mathsf{thistles},\\ &\mathsf{isn't\text{-}it\text{-}funny},\ \mathsf{cottleston\text{-}pie}\}\end{aligned}$$

The classes of these object identities are given by:

$$\begin{aligned}&\mathcal{I}(\mathsf{Piglet}) = \mathcal{I}(\mathsf{Eeyore}) = \texttt{Animal}\\ &\mathcal{I}(\mathsf{Pooh}) = \texttt{Bear}\\ &\mathcal{I}(\mathsf{Pooh\text{-}house}) = \mathcal{I}(\mathsf{Piglet\text{-}house}) = \mathcal{I}(\mathsf{Eeyore\text{-}house}) = \texttt{House}\\ &\mathcal{I}(\mathsf{hunny}) = \mathcal{I}(\mathsf{haycorns}) = \mathcal{I}(\mathsf{thistles}) = \mathcal{I}(\mathsf{condensed\text{-}milk}) = \texttt{Food}\\ &\mathcal{I}(\mathsf{isn't\text{-}it\text{-}funny}) = \mathcal{I}(\mathsf{cottleston\text{-}pie}) = \texttt{Poem}\end{aligned}$$

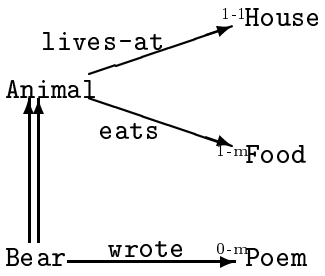


Figure 6: A simple example

and our attribute relations are

$$\begin{aligned}\mathcal{R} \;=\; \{&\mathsf{Pooh} \xrightarrow{\texttt{lives-at}} \mathsf{Pooh\text{-}house},\ \mathsf{Piglet} \xrightarrow{\texttt{lives-at}} \mathsf{Piglet\text{-}house},\\ &\mathsf{Eeyore} \xrightarrow{\texttt{lives-at}} \mathsf{Eeyore\text{-}house},\ \mathsf{Pooh} \xrightarrow{\texttt{eats}} \mathsf{hunny},\\ &\mathsf{Pooh} \xrightarrow{\texttt{eats}} \mathsf{condensed\text{-}milk},\ \mathsf{Piglet} \xrightarrow{\texttt{eats}} \mathsf{haycorns},\ \mathsf{Eeyore} \xrightarrow{\texttt{eats}} \mathsf{thistles},\\ &\mathsf{Pooh} \xrightarrow{\texttt{wrote}} \mathsf{isn't\text{-}it\text{-}funny},\ \mathsf{Pooh} \xrightarrow{\texttt{wrote}} \mathsf{cottleston\text{-}pie}\}\end{aligned}$$

Now we have to assign printable values to our object identities. The animals and the foods are easy:

$$\begin{aligned}\mathcal{V}(\mathsf{Pooh}) &= \text{“Winnie-the-Pooh”}\\ \mathcal{V}(\mathsf{Piglet}) &= \text{“Piglet”}\\ \mathcal{V}(\mathsf{Eeyore}) &= \text{“Eeyore”}\\ \mathcal{V}(\mathsf{hunny}) &= \text{“hunny”}\\ \mathcal{V}(\mathsf{haycorns}) &= \text{“haycorns”}\\ \mathcal{V}(\mathsf{thistles}) &= \text{“thistles”}\end{aligned}$$

The values for houses are a little more difficult to figure out. In particular we're not very sure of Pooh's address, though we do know that he lives under the name of *Sanders*, so that'll have to do:

$$\begin{aligned}\mathcal{V}(\mathsf{Pooh\text{-}house}) &= \text{“Sanders”}\\ \mathcal{V}(\mathsf{Piglet\text{-}house}) &= \text{“The Beach Tree”}\\ \mathcal{V}(\mathsf{Eeyore\text{-}house}) &= \text{“The House at Pooh Corner”}\end{aligned}$$

and finally, the poems:

$$
\begin{array}{rcl}
\mathcal{V}(\textsf{isn't-it-funny}) & = & \text{“Isn't it funny,} \\
 & & \text{How a bear likes honey?} \\
 & & \text{Buzz! Buzz! Buzz!} \\
 & & \text{I wonder why he does?”} \\
\mathcal{V}(\textsf{cottleston-pie}) & = & \text{“Cottleston, Cottleston, Cottleston Pie.} \\
 & & \text{A fly can't bird, but a bird can fly.} \\
 & & \text{Ask me a riddle and I reply,} \\
 & & \text{`}\textit{Cottleston, Cottleston, Cottleston Pie.}\text{'} \text{ ”} \\
 & & \ldots.
\end{array}
$$

Note that the arity constraints are satisfied: each animal lives at exactly one house, and eats at least one food.

3 Modeling Observations of Databases

The model for databases presented in Section 2 represents the information stored in a database and its structure. However, in practice, this information is not directly observable. Information on the contents of a database is obtained by querying the database and examining the results of the query. In this section we will develop a way of modeling observations of databases, and develop a correspondence between these observations and the internal representation based model developed in Section 2. We then go on to develop a theory about what sets of observations are possible, insofar as they may correspond to, and be generated by, some database model.

The concept of observations gives us an alternative kind of semantics for databases: the semantics of a database being the set of all possible observations for that database. We shall see in Section 4 that such a semantics would identify database instances which had different internal structures as represented by the model of Section 2.2. While, on the one hand, we might claim that database instances should be thought to be equivalent if they cannot be distinguished through querying, we might also postulate that two databases might behave differently as the result of various updates and manipulations even if they are observationally equivalent.

3.1 Basic Queries

We will model basic queries to a database by patterns, which we will call *types*, which are matched against the database. We do not attempt to model the more sophisticated queries, involving such things as predicates and quantification, that would be submitted to a real life database system via a high level query language, such as SQL. Such queries could be interpreted as generating a set of the simple queries described here, and then filtering the results in some way. However since such an interpretation would be dependent on the particular DBMS being considered, and since it would not be relevant to the issues being considered here, we do not attempt to address this problem in this paper.

Types

Intuitively a type is a pattern which is matched against a database. When such a pattern is submitted to a database as a query, the database responds with the set of the projections onto the pattern of all the objects in the database matching the pattern (a formal definition of "projections" will be given later in Section 3.2).

A *type* consists of a digraph and a distinguished node, called the *root*, such that there is a path from the root to every vertex in the digraph, and, in addition, the arrow relation of the digraph is functional in it's first two arguments. The vertices of the digraphs are either classes, which must match the same classes in the database schema, or *variables* which may match any class in the database schema, and the arrow labels are taken from the same set as the arrow labels of the database. Types are similar to the *regular trees* described in [Ait84] except that it is not possible to have several distinct vertices with the same labels.

Suppose $\mathcal{N}$ and $\mathcal{L}$ are sets of class names and arrow labels respectively, and X is a countably infinite set of **variables**. Then, formally, a **type** is a triple of the form $t = (V_t, E_t, r_t)$, where (V_t, E_t) is a digraph over $\mathcal{N} \cup X$ and $\mathcal{L}$, and $r_t \in V_t$, such that

T1 $\forall u, v, w \in V_t \cdot \forall a \in \mathcal{L} \cdot u \xrightarrow{a}_t v \ \wedge \ u \xrightarrow{a}_t w$ implies $v \equiv w$

T2 $\forall u \in V_t \cdot \exists a_1, \ldots, a_n \in \mathcal{L} \cdot \exists v_1, \ldots, v_{n-1} \in V_t \cdot r_t \xrightarrow{a_1}_t v_1 \xrightarrow{a_2}_t \ \ldots \ \xrightarrow{a_{n-1}}_t v_{n-1} \xrightarrow{a_n}_t u$

(where $u \xrightarrow{a}_t v$ means $(u, a, v) \in \mathcal{E}_t$). r_t is said to be the **root** of t.

We define the function *Root* from types to $\mathcal{N} \cup X$ such that $Root(t)$ is the root of t.

Some examples of these simple queries, together with their corresponding types (patterns, rooted digraphs or whatever), are given in Figure 7. The queries are expressed in a format which it is hoped will not require any explanation.

We will need to define some more apparatus for dealing with types.

We define the function Atr from types to sets of arrow labels by:

$$Atr(t) \ = \ \{a \in \mathcal{L} \mid \exists u \in V_t \cdot r_t \xrightarrow{a}_t u\}$$

where $t = (V_t, E_t, r_t)$. That is $Atr(t)$ is the set of all a such that r_t has an a-arrow in (V_t, E_t).

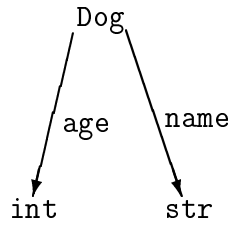
If $t = (V_t, E_t, r_t)$ is a type and $a \in Atr(t)$, then we write **t(a)** for the largest type $s = (V_s, E_s, r_s)$ such that (V_s, E_s) is a subgraph of (V_t, E_t) and $r_t \xrightarrow{a}_t r_s$. So $t(a)$ is the type formed by going down the a-arrow from the root of t, making the new vertex the root of the type, and forming the digraph of the type by removing any vertices of the digraph of t to which there are no paths from the new root of the type.

A type t is said to be a **ground type** if it does not involve any variables.

A **substitution** is a partial function, $\sigma : X \xrightarrow{\sim} \mathcal{C} \cup X$, with a finite domain of definition.

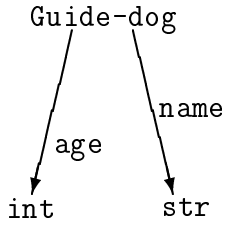
We extend substitutions to types in the obvious way. That is, for any type t, $\sigma(t)$ is the result of replacing all occurrences of x in t by $\sigma(x)$ for every $x \in \mathrm{dom}(\sigma)$.

(a)



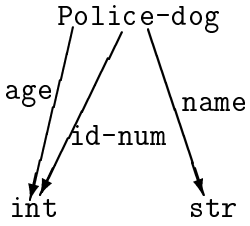
Dog[age:int,name:str]

(b)



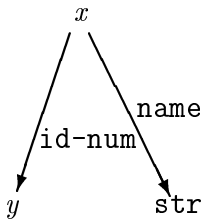
Guide-dog[age:int,name:str]

(c)



Police-dog[age:int,
name:str,id-num:int]

(d)



[id-num,name:str]

Figure 7: Some types

A *ground type* $t = (V_t, E_t, r_t)$ is said to be **valid** for a schema $\mathcal{G} = (\mathcal{C}, \mathcal{E}, \mathcal{S}, D^{\mathcal{C}})$ iff

1. $r_t \in \mathcal{C}$
2. For each $a \in Atr(t)$, if $r_t \xrightarrow{a}_t p$ then there is a $q \in \mathcal{C}$ such that $r_t \xrightarrow{a}_{\mathcal{G}} q$ and $p \Longrightarrow q$.
3. For each $a \in Atr(t)$ $t(a)$ is a valid ground type for $\mathcal{G}$.

A type t is said to be a **valid** type for a schema $\mathcal{G}$ iff there exists a substitution σ such that $\sigma(t)$ is a *valid ground type* for $\mathcal{G}$.

If $p \in \mathcal{C}$ is a class for a database schema $\mathcal{G} = (\mathcal{C}, \mathcal{E}, \mathcal{S}, D^{\mathcal{C}})$, then we write $\overline{p}$ for the *largest* valid ground type of the form $\overline{p} = (V, E, p)$ (that is a $\overline{p}$ is a ground type rooted at p). For

example, for the database schema shown in Figure 4, $\overline{\texttt{Dog}}$ is the type shown in Figure 7(a), while $\overline{\texttt{Police-dog}}$ is shown in Figure 7(c).

Subtype Relation

We define a "subtype" relation on valid types of a schema in order to capture the concept of one type being *more specific* than another type. By "more specific" what we mean is that every instance of the first type can also be considered to be an instance of the second type (or, rather, can be projected onto an instance of the second type). This is true if and only if the second type matches (can be superimposed on) every subtree of the database schema that the first type matches.

We will define our subtype relation in two stages, first defining the *simple subtype* relation.

If $t = (V_t, E_t, r_t)$ and $s = (V_s, E_s, r_s)$ are valid types for some schema, $\mathcal{G}$, then we say that t is a **simple subtype** of s for the schema $\mathcal{G}$, written $t \preceq_s s$, iff

1. $r_s \in X \ \vee \ r_t \Longrightarrow r_s$.
2. $\forall a \in Atr(s) \cdot t(a) \preceq_s s(a)$

That is, either r_s is a variable or r_t is a specialisation of r_s, and, for every attribute a of s, a is an attribute of t and $t(a)$ is a simple subtype of $s(a)$. Note that the relation $\preceq_s$ is reflexive and transitive, and, if there is a substitution, σ, such that $\sigma(s) = t$, then $t \preceq_s s$.

We say that t is a **subtype** of s, written $t \preceq s$, iff

Either $t \preceq_s s$

or $\exists a \in Atr(t) \cdot t(a) \preceq s$

That is, t is a subtype of s iff there is a subgraph of t which is a simple subtype of s. Note that $\preceq$ is a reflexive, transitive relation, and, if we equate types which differ only in the names of their variables (they are alpha-convertible) then it is anti-symmetric.

For example consider the types shown in figure 7 for the database schema shown in figure 4. The type shown Figure 7(c) is a subtype of the types shown in Figures 7(a) and 7(d), while the type in Figure 7(b) is a subtype of the one in 7(a) only.

3.2 Basic Observations

In this section we will describe a model for "basic observations" which are observations resulting from a single query of a database. In Section 3.3 we will go on to model "general observations" which can be made by submitting a number of queries to a database.

Domains of Atomic Observations

We will define the *domain* of a database schema to be the set of all atomic pieces of data, or records, that could be observed for a database satisfying the schema. Further, we will define the *domain* of a type to be the set of all atomic pieces of data that could be observed as the result of submitting the type to a database as a query.

Suppose we have a database schema $\mathcal{G}$ over class names $\mathcal{N}$ and arrow labels $\mathcal{L}$, and with printable values set $\mathcal{D}$. Then the **domain** of $\mathcal{G}$, $D_{\mathcal{G}}$, is the set of all partial functions f from strings of attributes from $\mathcal{L}$ to $\mathcal{D}$ such that the domain of f is closed under prefixing. That is:

$$D_{\mathcal{G}} \;=\; \{f : \mathcal{L}^+ \xrightarrow{\sim} \mathcal{D} \mid \gamma \in \mathrm{dom}(f) \wedge \gamma = \alpha\beta \;\text{ implies }\; \alpha \in \mathrm{dom}(f)\}$$

(where X^+ denotes the set of strings of elements of X).

For any partial function $f \in D_{\mathcal{G}}$ and any $a \in \mathcal{L}$, we denote by $f{\downarrow_a}$ the partial function defined by

$$f{\downarrow_a}\,(\gamma) \;=\; f(a\gamma)$$

for all $\gamma \in \mathcal{L}^+$. We use ϵ to denote the unique string of length 0.

For any type t the **language** of t, $Lan(t) \subseteq \mathcal{L}^+$, is the set of strings of arrow labels defined by:

$$Lan(t) \;=\; \{\epsilon\} \cup \{a\gamma \mid a \in Atr(t) \;\wedge\; \gamma \in Lan(t(a))\}$$

Note that, if $t \preceq_s s$, then $Lan(s) \subseteq Lan(t)$.

For any type, t, and any string of arrow labels $\gamma = a_1 \ldots a_k \in Lan(t)$, we write $t(\gamma)$ for the type $t(a_1)\ldots(a_k)$.

For any type, t, such that t is valid for $\mathcal{G}$, we define the define the **domain** of t for $\mathcal{G}$, $D_t \subseteq D_{\mathcal{G}}$, to be the set of all partial functions $f \in D_{\mathcal{G}}$ such that:

1. $dom(f) = Lan(t)$
2. For each $\gamma \in Lan(t)$, if $Root(t(\gamma)) = p \in \mathcal{C}$ then $f(\gamma) \in D^p$.

Note that, if $\gamma \in Lan(t)$ and $Root(t(\gamma)) = x \in X$, then $f(\gamma)$ must be defined but may take any value in $\mathcal{D}$.

The idea here is that D_t is the set of the possible values returned by a database in response to the type t being submitted as a query.

Note that, for any substitution σ, if $s = \sigma(t)$ then $D_s \subseteq D_t$.

Projections

We will define *projections* as mappings from the domain of one type to the domain of another, representing the *restrictions* of the instances of the first type to the second. Projections will be used later, in our definition of *observations*, in order to build an information ordering on observations.

In order for a projection from a type t to a type s to make sense, every instance of type t must give rise to an instance of type s in a natural way, and so every subtree of the database schema that t can match must be matched by s as well. Hence we only define projections for pairs of types t and s such that $t \preceq s$.

Our definition of projections is stratified: that we first give a recursive definition of *simple projections* and then define (general) projections as an extension of simple projections.

Suppose t and s are valid types such that $t \preceq_s s$. The (unique) **simple projection** from t to s is the function $\Psi : D_t \to D_s$ defined by:

$$\Psi(f)(\gamma) \;=\; f(\gamma)$$

for any $f \in D_t$ and any $\gamma \in Lan(s)$, and $\Psi(f)(\gamma)$ is undefined if $\gamma \notin Lan(s)$. If $t \not\preceq_s s$ then there is no simple projection from t to s.

A (general) **projection** from t to s is a function $\Pi : D_t \to D_s$ constructed using the following rules:

1. If Π is a simple projection from t to s then Π is a projection from t to s.
2. If, for some $a \in Atr(t)$, $t(a) \preceq s$ and Π' is a projection from $t(a)$ to s, then if

   $$\Pi(f) = \Pi'(f{\downarrow_a})$$

   for all $f \in D_t$, then Π is a projection from t to s.

Note that, if $r \preceq s$ and $s \preceq t$, and Π and Π' are projections from r to s and s to t respectively, then $\Pi' \circ \Pi$ is a projection from r to t.

Some examples of projections, for some types of the database schema shown in figure 2, are depicted in Figure 8. It is worth noting that, as in the case of the types shown in Figure 8(c), there may be several different projections between two types.

Bags

Due to our notion of object identity, it will be possible for a query to give rise to several indistinguishable pieces of atomic information, and so wo will need to make use of constructs which allow for multiplicity of objects in our representation of basic observations. We introduce *bags* or *multisets* for this purpose. Bags are similar to sets except that they allow multiple copies of elements.

A **bag** over a set D is a function $u : D \to \mathbb{N}$ such that $u(x) = 0$ for all but finitely many $x \in D$. The idea is that, for each element $x \in D$, $u(x)$ is the multiplicity of x in the collection of elements of D represented by u. We write $D^\star$ for the set of all bags over D.

In accordance with the idea of bags representing collections of elements, we use the notation $[x_1, \ldots, x_n]$ to denote the bag with elements $x_1, \ldots, x_n$ (counting multiplicitly). Formally

$$\begin{aligned} [](x) &= 0 \\ [x_1, \ldots, x_{n+1}](x) &= \begin{cases} [x_1, \ldots, x_n](x) + 1 & \text{if } x = x_{n+1} \\ [x_1, \ldots, x_n](x) & \text{otherwise} \end{cases} \end{aligned}$$

for any $x \in D$. We use this notation interchangeably with the function notation for bags.

Note that, for any $x_1, \ldots, x_n$,

$$[x_1, \ldots, x_n] = [x_{i_1}, \ldots, x_{i_n}]$$

where $i_1, \ldots, i_n$ is any permutation of $1, \ldots, n$.

We say that x is a member of a bag u, and write $x \in u$, iff $u(x) \geq 1$.

We say a bag u is a *sub-bag* of a bag v, and write $u \subseteq v$, if and only if v consists of all the elements of u plus some additional elements. Hence $u \subseteq u$ iff

$$\forall x \in D \cdot u(x) \leq v(x)$$

If f is a function from D to E then we define the function $f^\star$, mapping $D^\star$ to $E^\star$, by:

$$f^\star(u)(y) \;=\; \sum\{u(x) \mid f(x) = y\}$$

for all $y \in E$. Note that, for any $x_1, \ldots, x_n \in D$,

$$f^\star([x_1, \ldots, x_n]) = [f(x_1), \ldots, f(x_n)]$$

We define the operators *union* and *merge* on bags as follows:

For any bags u and v, over D,

$$(u \cup v)(x) = u(x) + v(x)$$

for all $x \in D$

For any bags u and v, over D,

$$\mathit{Merge}(u, v)(x) = \mathit{max}(u(x), v(x))$$

for all $x \in D$.

Both $\cup$ and *Merge* are associative operators on bags and hence can be extended to finite sets of bags in the obvious way.

We define the operator *squash* on bags, which removes multiplicity of elements, by:

$$\mathit{Squash}(u)(x) \;=\; \min(1, u(x))$$

for any bag u over D and any $x \in D$.

Basic Observations

Basic observations are used to model information gained from a single query of a database. In section 3.3 they will be used to form a basis for a lattice of more general observations, representing the results of any number of queries.

A **basic observation** for a database schema, $\mathcal{G}$, consists of a pair,

$$\langle [x_1, \ldots, x_n], t \rangle$$

where t is a valid type for $\mathcal{G}$ and $[x_1, \ldots, x_n] \in D_t^\star$ is a bag of elements from the domain of t.

Such a basic observation is intended to represent the fact that, when the type t is submitted to a database as a query, the database responds with a set of tuples including ones corresponding to the elements $x_1, \ldots, x_n$ of D_t (counting multiplicity).

Basic Observations as a Poset

Suppose that BO is the set of all basic observations for some database schema $\mathcal{G}$. We will introduce a relationship $\vdash$ on BO, the idea of which is that, if $\langle x, t\rangle \vdash \langle y, s\rangle$, then for any database for which the observation $\langle x, t\rangle$ can be made, the observation $\langle y, s\rangle$ can also be made.

We define the relation $\vdash$ on BO to be the smallest pre-order (transitive, reflexive relation) satisfying:

1. If $t \preceq s$ and $u \in D^\star_t$, $v \in D^\star_s$ are such that, for some projection Π from t to s,

   $$v \subseteq \mathit{Squash}(\Pi^\star(u))$$

   then $\langle u, t\rangle \ \vdash \ \langle v, s\rangle$.

2. If $t \preceq_s s$, $u \in D^\star_t$, $v \in D^\star_s$ and Ψ is the unique simple projection from t to s, then if

   $$v \subseteq \Psi^\star(u)$$

   then $\langle u, t\rangle \ \vdash \ \langle v, s\rangle$.

3. If $t \preceq s$ and $u \in D^\star_t$, $v \in D^\star_s$ are such that there is a collection of bags, $\{v_i \in D^\star_s \mid i \in I\}$, such that

   $$v \ \subseteq \ \mathit{Merge}_{i\in I}(v_i)$$

   and, for all $i \in I$,

   $$\langle u, t\rangle \ \vdash \ \langle v_i, s\rangle$$

   then $\langle u, t\rangle \ \vdash \ \langle v, s\rangle$.

I other words $\langle u, t\rangle \vdash \langle v, s\rangle$ if t is a subtype of s, for every element $x \in v$ there is an element $y \in u$ and a projection Π from t to s such that $x = \Pi(y)$, and, if the multiplicity of x in v is greater than 1, then $t \preceq_s s$ and each occurrence of x in u arises from a distinct element $y \in v$ via the simple projection Ψ from t to s. The reason for this somewhat complex definition of $\vdash$ is that, while we expect simple projections to preserve multiplicity of objects, general projections may project several objects onto the same object in the database. so that we must be careful to *squash* multiple occurrences of a tuple arising through non-simple projections into a single occurrence of the tuple.

Unfortunately $\vdash$ is not a *partial order* as we would like, since it fails to be anti-symmetric. Consequently we will consider the *poset* **BO** generated by BO and $\vdash$.

We will say that a basic observation $\langle u, t\rangle$ for a database schema $\mathcal{G}$ is **canonical** if, for any other basic observation $\langle v, t\rangle$ such that $\langle v, t\rangle \vdash \langle u, t\rangle$ and $\langle u, t\rangle \vdash \langle v, t\rangle$, we have $v \subseteq u$. That is $\langle u, t\rangle$ is the largest basic observation conveying that information.

If we write BO' for the set of canonical basic observations for a schema $\mathcal{G}$, then we find that $(BO', \vdash)$ is a poset and is isomorphic to **BO**. Consequently we will represent the equivalence classes in **BO** by the canonical basic observations that they contain.

3.3 General Observations

We will define *general observations* to represent the result of submitting a number of queries to a database and observing the results.

We define a **general observation** for a database schema $\mathcal{G}$ to be a set of basic observations for $\mathcal{G}$. However, since there may be many subsets of the basic observations with equivalent informational content, and since we wish to define a lattice of general observations, we will need to find a class of *canonical general observations*, as we did for basic observations, and restrict our attention to these.

Consider a database schema $\mathcal{G}$ with corresponding basic observations **BO**. We start by extending the relation $\vdash$ to a relation between sets of basic observations and individual basic observations:

> Suppose $X \subseteq \mathbf{BO}$ and $\langle u, t\rangle \in \mathbf{BO}$. Then $X \vdash \langle u, t\rangle$ iff there exists a collection of bags, $\{u_i \in D_t^\star \mid i \in I\}$, and of basic observations, $\{p_i \in X \mid i \in I\}$, such that
>
> $$u \subseteq \text{Merge}_{i\in I}(u_i)$$
>
> and, for all $i \in I$,
>
> $$p_i \vdash \langle u_i, t\rangle$$

The idea here is that, if $X \vdash \langle u, t\rangle$ and all the basic observations in X are observable for some database, then $\langle u, t\rangle$ is also observable.

Next we overload the symbol $\vdash$ yet again by extending the above relation to a relation between sets of basic observations:

> Suppose $X \subseteq \mathbf{BO}$ and $Y \subseteq \mathbf{BO}$. Then $X \vdash Y$ iff
>
> $$\forall q \in Y \cdot X \vdash q$$

So, for any database in which all the basic observations in X are observable, all the observations in Y are also observable.

It is clear that the relation $\vdash$ on $\mathcal{P}(BO)$ is transitive and reflexive. Unfortunately it is not anti-symmetric.

We say that a general observation, $X \subseteq \mathbf{BO}$, is **canonical** iff, for any $Y \subseteq \mathbf{BO}$ such that $Y \vdash X$ and $X \vdash Y$, we have $Y \subseteq X$.

For any $X \subseteq \mathbf{BO}$ we can construct a canonical general observation $\uparrow X$ such that $X \vdash \uparrow X$ and $\uparrow X \vdash X$ by:

$$\uparrow X \ = \ \{p \in \mathbf{BO} \mid X \vdash p\}$$

So our canonical general observations are exactly the upwards closed subsets of **BO** under the ordering. Suppose that GO is the set of all such upwards closed sets of basic observations for a schema $\mathcal{G}$.

Proposition 3.1 *For any $X, Y \in GO$, $X \vdash Y$ iff $Y \subseteq X$.*

Proof: trivial ∎

We define the operators $\sqcup$ and $\sqcap$ on GO by

$$X \sqcup Y \ = \ X \cap Y$$

and

$$X \sqcap Y \ = \ X \cup Y$$

Proposition 3.2 $(GO, \vdash, \sqcup, \sqcap)$ *is a lattice.*

Proof: trivial ∎

So we define our lattice of **general observations**, **GO**, for a database schema $\mathcal{G}$ by:

$$\mathbf{GO} = (GO, \vdash, \sqcup, \sqcap)$$

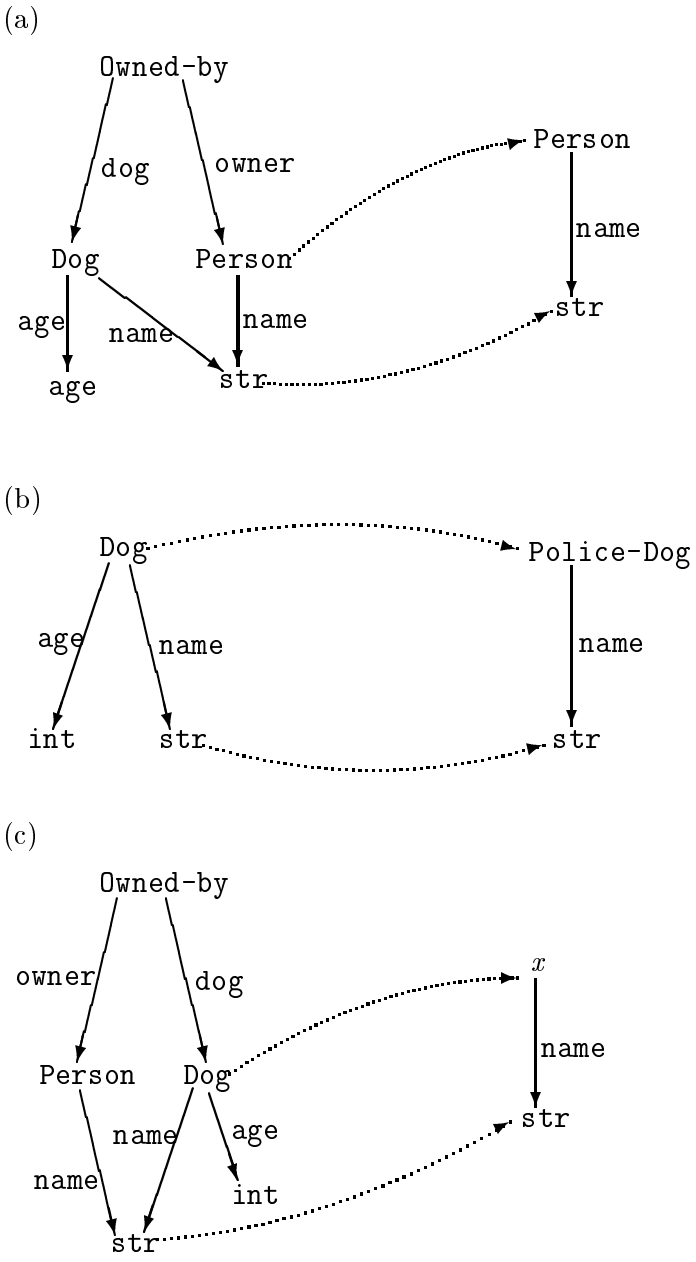


Figure 8: Some examples of projections

4 The Correspondence of Models and Observations

In Section 2 we defined models to represent the information stored in a database, and in Section 3 we defined a representation for the observations that can be made by querying a database. In this section we will develop a correspondence between these two concepts, and, in particular, will derive a relation showing what sets of relations can be made for a particular model of a database instance.

Suppose that we have a model $\mathcal{M} = (\mathcal{O}, \mathcal{R}, \mathcal{I}, \mathcal{V})$ such that $\mathcal{M}$ satisfies a database schema $\mathcal{G}$ over class names $\mathcal{N}$ and arrow labels $\mathcal{L}$.

For any $o \in \mathcal{O}$ and any valid type $t = (V_t, E_t, r_t)$ (for schema $\mathcal{G}$ and variables X), we say that o *is of type* t, and write $\mathcal{M} \rhd o : t$, iff

1. Either $r_t \in X$ or $r_t \in \mathcal{I}(o)$.
2. For each $a \in Atr(t)$ there exists an $o' \in \mathcal{O}$ such that $o \xrightarrow{a}_{\mathcal{M}} o'$ and $\mathcal{M} \rhd o' : t(a)$.

The idea here is that the type t *matches* the object with object identity o. That is, o has a class which matches the root of t, and for each arrow label $a \in Atr(t)$, o has an a-arrow with type $t(a)$.

For any object identity $o \in \mathcal{O}$ and any valid type t such that $\mathcal{M} \rhd o : t$, we define $[\![\mathcal{M} \rhd o : t]\!] \in D_{\mathcal{G}}$ by:

1. If ϵ is the unique string of length 0, then
$$[\![\mathcal{M} \rhd o : t]\!](\epsilon) \;=\; \mathcal{V}(o)$$
2. If $a \in Atr(t)$ and $o \xrightarrow{a}_{\mathcal{M}} o'$ then for any string $\gamma \in \mathcal{L}^+$
$$[\![\mathcal{M} \rhd o : t]\!](a\gamma) \;=\; [\![\mathcal{M} \rhd o' : t(a)]\!](\gamma)$$
3. For any other string $\gamma \in \mathcal{L}^+$, $[\![\mathcal{M} \rhd o : t]\!](\gamma)$ is undefined.

The idea is that $[\![\mathcal{M} \rhd o : t]\!]$ is the *value* or *meaning* of o in $\mathcal{M}$ projected onto the type t.

Proposition 4.1 *For any valid type t, and any $o \in \mathcal{O}$ of type t, $[\![\mathcal{M} \rhd o : t]\!] \in D_t$.*

Proof: It is enough to show that, for any string of arrow labels $\gamma \in \mathcal{L}^+$, $[\![\mathcal{M} \rhd o : t]\!](\gamma)$ is defined iff $\gamma \in Lan(t)$, and, if it is defined then it is in the required subset of $\mathcal{D}$. Proof is by induction on the length of γ. ∎

Proposition 4.2 *Suppose that s and t are valid types such that $t \preceq s$. Then:*

1. *If Ψ is a simple projection from t to s, and*

$$[\![\mathcal{M} \rhd o : t]\!] = x$$

for some $o \in \mathcal{O}$. Then

$$[\![\mathcal{M} \rhd o : s]\!] \;=\; \Psi(x)$$

2. *If Π is a projection from t to s, and*

$$[\![\mathcal{M} \rhd o : t]\!] = x$$

for some $o \in \mathcal{O}$. Then there exists an $o' \in \mathcal{O}$ such that

$$[\![\mathcal{M} \rhd o' : s]\!] \;=\; \Pi(x)$$

Proof: For part 1 it is sufficient to show that, for any $\gamma \in Lan(s)$,

$$[\![\mathcal{M} \rhd o : t]\!](\gamma) = [\![\mathcal{M} \rhd o : s]\!](\gamma)$$

which can be proved by induction on the length of γ. It follows from proposition 4.1 that $[\![\mathcal{M} \rhd o : s]\!](\gamma)$ is undefined for all $\gamma \notin Lan(s)$.

Part 2 is proved by induction on the definition of general projections, Π, Part 1 providing the base case. ∎

Suppose $\langle u, t\rangle \in \mathbf{BO}$ is a basic observation for the schema $\mathcal{G}$. Then we say we say $\langle u, t\rangle$ is **observable** in the model $\mathcal{M}$, and write

$$\mathcal{M} \models \langle u, t\rangle$$

iff, for each $x \in u$ such that $u(x) = k$ (the multiplicity of x in u is k) there exist *distinct* object identities $o_1, \ldots, o_k \in \mathcal{O}$ such that

$$[\![\mathcal{M} \rhd o_i : t]\!] \;=\; x$$

for $i = 1, \ldots, k$.

Proposition 4.3 *If* $\langle u,t\rangle, \langle v,s\rangle \in \mathbf{BO}$ *are basic observations such that* $\langle u,t\rangle \vdash \langle v,s\rangle$ *then*

$$(\mathcal{M} \models \langle u,t\rangle) \text{ implies } (\mathcal{M} \models \langle v,t\rangle)$$

Proof: We can prove that the property is preserved by each of the rules in the definition of the relation $\vdash$. That is we prove that

1. If $t \preceq_s s$ and $v \subseteq \Psi^\star(u)$, where Ψ is the unique simple projection from t to s, then

$$(\mathcal{M} \models \langle u,t\rangle) \text{ implies } (\mathcal{M} \models \langle v,t\rangle)$$

2. If Π is a projection from t to s, and $v \subseteq \mathit{Squash}(\Pi^\star(u))$, then

$$(\mathcal{M} \models \langle u,t\rangle) \text{ implies } (\mathcal{M} \models \langle v,t\rangle)$$

3. If $\{u_i \mid i \in I\}$ are such that

$$u \subseteq \mathit{Merge}_{i\in I}(u_i)$$

and, for each $i \in I$, $\mathcal{M} \models \langle u_i,t\rangle$ then $\mathcal{M} \models \langle u,t\rangle$.

It follows from the transitivity of logical implication that the property holds for the transitive closure of the rules. ∎

For any general observation $X \in \mathbf{GO}$ for $\mathcal{G}$ we say that X is *observable* in a model $\mathcal{M}$, and write $\mathcal{M} \models X$, iff

$$\forall \langle u,t\rangle \in X \cdot \mathcal{M} \models \langle u,t\rangle$$

Theorem 4.4 *If* $X, Y \in \mathbf{GO}$ *are general observations, such that* $X \vdash Y$*, then*

$$(\mathcal{M} \models X) \text{ implies } (\mathcal{M} \models Y)$$

Proof: Omitted. ∎

Theorem 4.5 *If* $X \in \mathbf{GO}$*, where* **GO** *is the lattice of general observations for a database schema* $\mathcal{G}$*, and every type in* X *is a ground type (that is, for each* $\langle u,t\rangle \in X$ t *is a valid ground type for* $\mathcal{G}$*), then there exists a model* $\mathcal{M}$*, satisfying* $\mathcal{G}$*, such that*

$$\mathcal{M} \models X$$

Proof: We will show how to construct a model $\mathcal{M}$ from X and leave the reader to satisfy himself that $\mathcal{M}$ satisfies $\mathcal{G}$ and $\mathcal{M} \models X$.

Note that the set of valid ground types for a schema $\mathcal{G}$ is finite, since $\mathcal{G}$ is a finite graph. Let $Types(X)$ be the set of types occurring in X.

For each $t \in Types(X)$ define $u_t \in D_t^\star$ by

$$u_t \;=\; Merge\{u \in D_t^\star \mid \langle u, t\rangle \in X\}$$

So that $\langle u_t, t\rangle$ is the largest basic observation with type t in X (note X is upwards closed under the ordering $\vdash$).

Next, for each type $t \in Types(X)$ define $v_t \in D_t^\star$ by

$$v_t \;=\; u_t - Merge\{\Psi_{s,t}(u_s) \mid s \in Types(X) \wedge s \preceq_s t\}$$

where $-$ denotes *bags difference* and $\Psi_{s,t}$ denotes the simple projection from s to t. Intuitively $\langle v_t, t\rangle$ denotes the largest basic observation in X not implied by some other basic observations in X (via simple projections).

Now we form $\mathcal{O}$, the set of object identities for our model as follows: for each type $t \in Types(X)$ and each $x \in v_t$, if $v_t(x) = k$ then we add k new object identities, $o_1^{t,x}, \ldots, o_k^{t,x}$, to $\mathcal{O}$. We then set

$$\begin{aligned} \mathcal{V}(o_i^{t,x}) &= x(\epsilon) \\ \mathcal{I}(o_i^{t,x}) &= \{p \in \mathcal{C} \mid Root(t) \Longrightarrow p\} \end{aligned}$$

Next we must form the arrow relation, $\mathcal{R}$, of $\mathcal{M}$. For each type $t \in Types(X)$ and each $x \in v_t$ there is a $u \in D_t^\star$ such that $\langle u, t\rangle \in X$ and $x \in u$. But, for each $a \in Atr(t)$, $\langle u, t\rangle \;\vdash\; \langle [x\!\downarrow a], t(a)\rangle$, and so $\langle [x\!\downarrow a], t(a)\rangle \in X$. Hence there is a type $s \in Types(X)$ such that $s \preceq_s t(a)$ and there is a $y \in v_s$ such that $\Psi_{s,t(a)}(y) = x\!\downarrow a$. So, for $i = 1, \ldots, v_t(x)$, we add the arrow

$$o_i^{t,x} \;\xrightarrow{a}_{\mathcal{M}}\; o_1^{s,y}$$

to $\mathcal{R}$.

Finally for any $t \in Types(X)$, if, for some arrow label a, $a \notin Atr$ but $Root(t)$ has an a-arrow of class q in $\mathcal{G}$, then, for each $x \in v_t$ and for $i = 1, \ldots, v_t(x)$, pick a random object identity, o, with class q (creating a new object identity if necessary, with printable value and arrows chosen arbitrarily) and add the arrow

$$o_i^{t,x} \;\xrightarrow{a}_{\mathcal{M}}\; o$$

to $\mathcal{R}$. ∎

5 Merging Databases

The problem of database merging 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, [SBD*81], [SL90]), 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 splits naturally into two parts: forming a common view (schema) for the databases, and combining the data contained in them.

The problem of combining data arising from a number of independent databases is a complex one, and involves resolving problems such as deciding when data items refer to a common object, deciding which version of some conflicting data to use, and so on. Many of these decisions are somewhat arbitrary and are dependent on the conceptual meanings and the origins of the databases involved. There is no single, inherently correct set of deciding principles which can be deduced from the database structures themselves. We will direct readers to [WHW90] for coverage of these aspects of database merging and will not consider them any further in this paper.

5.1 Merging Database Schemas

The rest of this section is dedicated to the problem of finding a common view of a number of distinct databases, that is finding a database schema that, in some sense, *supersedes* the schemas of the databases. Much has been written on the subject of merging database schemas (see [BLN86] 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, [Mot87, SBD*81], to algorithms which take two schemas, together with some constraints, and produce a merged schema [SLCN88]. 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. In this section we develop a semantic basis for expressing the merge of a collection of schemas in terms of the informational content of those schemas, together with a notion of consistency of the associated data.

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. To start with we will limit ourselves to considering only exact correspondences of database classes: that is the situation

where two or more classes correspond to exactly the same class of real world objects. The designer of the system is, therefore, called upon to resolve naming conflicts, whether homonyms or synonyms, by renaming classes and arrows where appropriate. 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. 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 introduced[1]. However, if the user intends them to be separate classes, the classes must be renamed (for example, *Dog1* and *Dog2*).

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 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. (Note that the definition in Section 2.2 of when a model *satisfies* a database schema allows the model to contain extra information which does not concern the classes or arrow relationships in the schema).

Our approach will be 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. We will hope to define such a binary merge as the join (least upper bound) of two schemas under some suitable information ordering.

Some problems with finding merges of schemas

Now that we have some intuitive idea of what the merge of two databases should be, we would like to find some formal definition of an information ordering on schemas and hence of merges of schemas. We can 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 (but then, isn't everything

[1] This contrasts with [SBD*81], which states that entities *should* should be renamed, and specialization edges introduced to *avoid* introducing null values

worthwhile?).

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 9 shows two

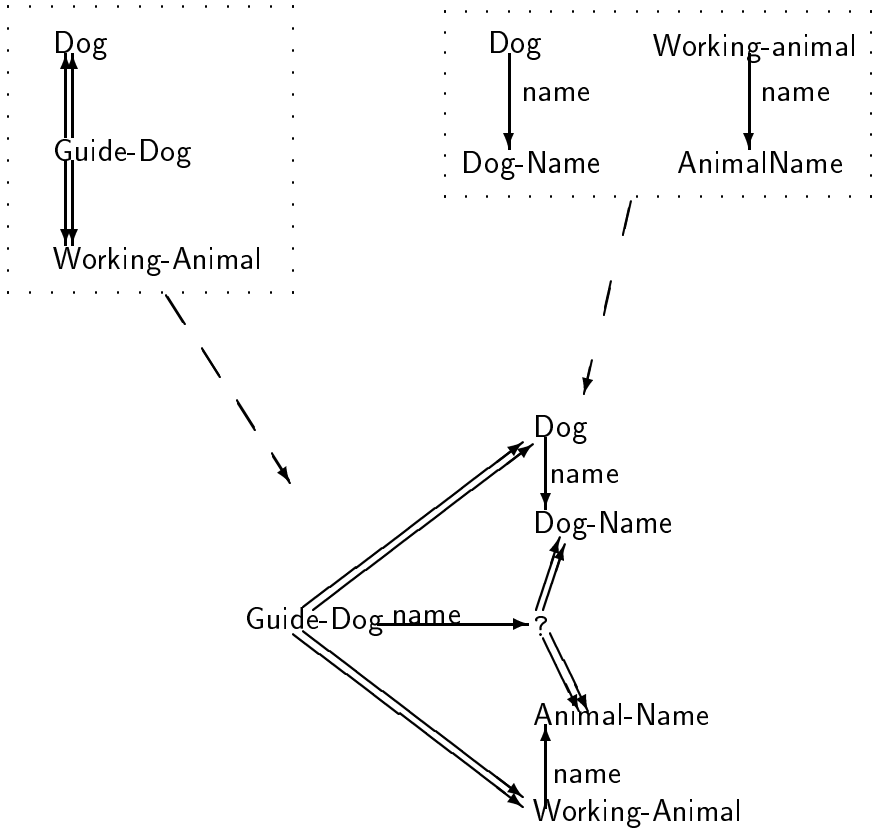


Figure 9: Schema merging involving implicit classes

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. For example consider

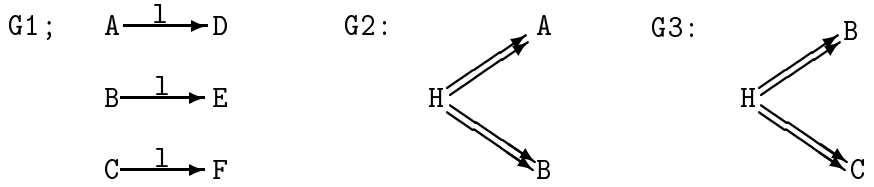


Figure 10: Some simple schemas

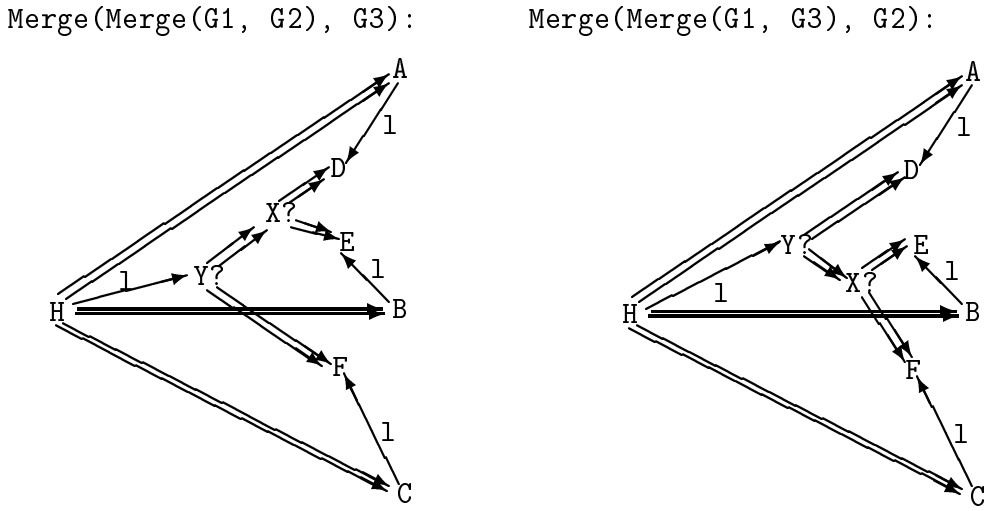


Figure 11: An example of non-associative merging

the three simple schemas shown in figure 10 (unfortunately these examples are getting too contrived for me to think of good examples involving dogs or Winnie-the-Pooh characters). If we were to first merge the schemas `G1` and `G2` we would need to introduce a new implicit class (`X?`) below `D` and `E`, and then merging with `G3` would make us introduce another new class below `X?` and `F`, yielding the first schema shown in figure 11. On the other hand, if we were to merge `G1` with `G3` and then merge the result with `G2`, we would first introduce an implicit

class below E and F, and then introduce another one below this one and D. Clearly what we really want is a single implicit class which is a specialisation of all three of D, E and F.

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 present all the information of another (plus

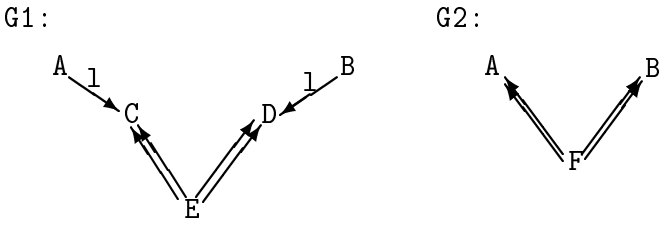


Figure 12: Yet more schemas

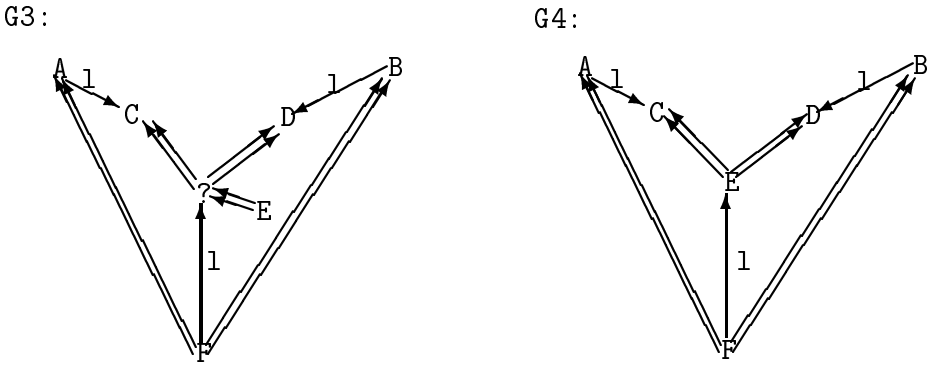


Figure 13: Possible candidates for the merges of the schemas

additional information) it must have, at least, all the normal classes of the other, however let us consider the two schemas shown in figure 12. We would like to assert that the schema G3 shown in figure 13 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 l-arrow of F has class E, which may impose 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 l-arrow of F has both classes C and D.

We could proceed by extending our definition of database schemas to allow special implicit

classes, and then try to find an ordering which takes account of them. However the complexities involved in such a treatment are quite great, and so we will adopt a slightly different approach.

5.2 Weak Schemas

In order to avoid the complexities of introducing implicit classes, what we will do is first weaken our definition of database schemas so that these classes become unnecessary (they are indeed *implicit*). We will then define an information ordering on these *weak schemas*, such that binary joins do exist and are associative. Finally we will present a method of converting a weak schema into a proper schema by introducing additional "implicit" classes. (In an attempt to avoid excessive confusion we will refer to the database schemas defined in section 2.1 as *proper schemas* in places where some ambiguity is likely to arise). The idea is that we can do all our merging using weak schemas, and then convert the result to a proper schemas when we are done.

A **weak schema** over class names $\mathcal{N}$ and arrow labels $\mathcal{L}$, with printable values $\mathcal{D}$, is a five-tuple $(\mathcal{C}, \mathcal{E}, \mathcal{S}, \mathcal{K}, \mathcal{D}^{\mathcal{C}})$, where $\mathcal{C} \subseteq \mathcal{N}$, $\mathcal{E} \subseteq \mathcal{C} \times \mathcal{L} \times \mathcal{C}$, $\mathcal{S} \subseteq \mathcal{C} \times \mathcal{C}$, $\mathcal{K} : \mathcal{E} \to \{0\text{-m},1\text{-m},0\text{-}1,1\text{-}1\}$ and $\mathcal{D}^{\mathcal{C}} : \mathcal{C} \to \mathcal{P}(\mathcal{D})$, such that

W1 If $p, q, r \in \mathcal{C}$ and $a \in \mathcal{L}$ are such that $p \xrightarrow{a} q$ and $p \xrightarrow{a} r$ and $q \Longrightarrow r$ then $q = r$.

W2 $\mathcal{S}$ is a partial order on $\mathcal{C}$.

W3 For any $p, q, r \in \mathcal{C}$ and $a \in \mathcal{L}$ such that $q \xrightarrow{a} r$ and $p \Longrightarrow q$ there exists an $s \in \mathcal{C}$ such that $p \xrightarrow{a} s$ and $s \Longrightarrow r$.

W4 For any $p, q, r \in \mathcal{C}$ and any $a \in \mathcal{L}$, if $p \xrightarrow{a} q$ and $p \xrightarrow{a} r$ then $\mathcal{K}(p \xrightarrow{a} q) = \mathcal{K}(p \xrightarrow{a} r)$.

W5 For any $p, q, r, s \in \mathcal{C}$ and any $a \in \mathcal{L}$, if $p \xrightarrow{a} q$ and $r \xrightarrow{a} s$ and $r \Longrightarrow p$ and $s \Longrightarrow q$, then $\mathcal{K}(p \xrightarrow{a} q) \leq \mathcal{K}(r \xrightarrow{a} s)$.

W6 For any $p, q \in \mathcal{C}$, if $p \Longrightarrow q$ then $\mathcal{D}^p \subseteq \mathcal{D}^q$.

(where $p \xrightarrow{a} q$ means $(p, a, q) \in \mathcal{E}$ and $p \Longrightarrow q$ means $(p, q) \in \mathcal{S}$ as with proper schemas).

The axiom **W4**, which says that for any class p all the a-arrows of p have the same arity, makes sense if we understand that when a class p has several a-arrows, say $p \xrightarrow{a} q_1, \ldots, p \xrightarrow{a} q_n$, it means that the a arrows of any instance of p must have all of the classes $q_1, \ldots, q_n$.

Note that any proper database schema, according to the definitions in section 2.1, is also a weak schema according to this definition. In fact weak schemas differ from proper schemas in the following ways:

1. We no longer require that $\mathcal{E}$ be functional in its first two arguments. Instead 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 $\mathcal{S}$.

2. We no longer require that $\mathcal{D}^{\mathcal{C}}(p) \neq \emptyset$ for every class $p \in \mathcal{C}$.

An ordering on weak schemas

We will now define an ordering $\sqsubseteq$ on weak schemas. Suppose $\mathcal{G}_1 = (\mathcal{C}_1, \mathcal{E}_1, \mathcal{S}_1, \mathcal{K}_1, \mathcal{D}_1)$ and $\mathcal{G}_2 = (\mathcal{C}_2, \mathcal{E}_2, \mathcal{S}_2, \mathcal{K}_2, \mathcal{D}_2)$ are weak schemas over $\mathcal{N}$ and $\mathcal{L}$. We write $\mathcal{G}_1 \sqsubseteq \mathcal{G}_2$ iff

1. $\mathcal{C}_1 \subseteq \mathcal{C}_2$

2. If $p, q \in \mathcal{C}_1$ and $a \in \mathcal{L}$ are such that $p \xrightarrow{a}_1 q$ then there exists an $r \in \mathcal{C}_2$ such that $p \xrightarrow{a}_2 r$ and $r \Longrightarrow_2 q$.

3. $\mathcal{S}_1 \subseteq \mathcal{S}_2$

4. If $p, q \in \mathcal{C}_1$,$r \in \mathcal{C}_2$ and $a \in \mathcal{L}$ are such that $p \xrightarrow{a}_1 q$, $p \xrightarrow{a}_2 r$ and $r \Longrightarrow q$ then $\mathcal{K}_1(p \xrightarrow{a}_1 q) \leq \mathcal{K}_2(p \xrightarrow{a}_2 r)$.

5. For each $p \in \mathcal{C}_1$, $\mathcal{D}_2(p) \subseteq \mathcal{D}_1(p)$.

(where we use the notation $p \xrightarrow{a}_i q$ to mean $(p, a, q) \in \mathcal{E}_i$, and $p \Longrightarrow_i q$ to mean $(p, q) \in \mathcal{S}_i$). The only one of these that is likely to require any additional justification is 2. It means that, for any class $p \in \mathcal{C}_1$, if p has an a-arrow of class q in $\mathcal{G}_1$, then p must also have an a-arrow in $\mathcal{G}_2$ with a class at least as specific as q and a cardinality constraint at least as specific as that of the a-arrow in $\mathcal{G}_1$.

We will leave the reader to convince himself that, if $\mathcal{G}_1 \sqsubseteq \mathcal{G}_2$ then instances $\mathcal{G}_1$ can be considered to be instances of $\mathcal{G}_2$ via a natural projection. This problem, while intuitively simple, is complicated by the fact that we haven't actually defined instances of weak schemas, and we would need to do so in order to handle this formally.

Proposition 5.1 *$\sqsubseteq$ is a partial ordering on weak schemas.*

Proof: Transitivity and reflexivity are straight forward. We will prove antisymmetry.

Suppose $\mathcal{G}_1$ and $\mathcal{G}_2$ are as above, and $\mathcal{G}_1 \sqsubseteq \mathcal{G}_2$ and $\mathcal{G}_2 \sqsubseteq \mathcal{G}_1$. Then $\mathcal{C}_1 \subseteq \mathcal{C}_2$ and $\mathcal{C}_2 \subseteq \mathcal{N}_1$ so $\mathcal{C}_1 = \mathcal{C}_2$. Simillarly we get $\mathcal{S}_1 = \mathcal{S}_2$ and, for every $p \in \mathcal{C}_1$, $\mathcal{D}_1(p) = \mathcal{D}_2(p)$. We need to show that, for any $p, q \in \mathcal{C}_1$ and any $a \in \mathcal{L}$, $p \xrightarrow{a}_1 q$ iff $p \xrightarrow{a}_2 q$.

Suppose $p \xrightarrow{a}_1 q$. Then, since $\mathcal{G}_1 \sqsubseteq \mathcal{G}_2$, there is an $r \in \mathcal{C}_2 (= \mathcal{C}_1)$ such that $p \xrightarrow{a}_2 r$ and $r \Longrightarrow_2 q$. Hence, since $\mathcal{G}_2 \sqsubseteq \mathcal{G}_1$, there is an $s \in \mathcal{C}_1$ such that $p \xrightarrow{a}_1 s$ and $s \Longrightarrow_1 r$. So we have $p \xrightarrow{a}_1 q$ and $p \xrightarrow{a}_1 s$ and $s \Longrightarrow_1 r \Longrightarrow_1 q$ (since $\mathcal{S}_1 = \mathcal{S}_2$), and so, from the definition of weak schemas, $s = q$. Hence $q \Longrightarrow_1 r \Longrightarrow_1 q$, so $q = r$ and $p \xrightarrow{a}_2 q$. Simillarly, if $p \xrightarrow{a}_2 q$ then $p \xrightarrow{a}_1 q$. Further, since $\mathcal{K}_1(p \xrightarrow{a}_1 q) \leq \mathcal{K}_2(p \xrightarrow{a}_2 q)$ and $\mathcal{K}_2(p \xrightarrow{a}_2 q) \leq \mathcal{K}_1(p \xrightarrow{a}_1 q)$, we have $\mathcal{K}_1(p \xrightarrow{a}_1 q) = \mathcal{K}_2(p \xrightarrow{a}_2 q)$. ∎

Proposition 5.2 *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 $\mathcal{G}_1 \sqcup \mathcal{G}_2$.*

Proof: Given weak schemas $\mathcal{G}_1$ and G_2 as above, define $\mathcal{G} = (\mathcal{C}, \mathcal{E}, \mathcal{S}, \mathcal{D}^{\mathcal{C}})$ by

$$\begin{aligned}
\mathcal{C} &= \mathcal{C}_1 \cup \mathcal{C}_2 \\
\mathcal{S} &= (\mathcal{S}_1 \cup \mathcal{S}_2)^* \\
\mathcal{E} &= \{(p,a,q) \mid (\exists r \in \mathcal{C} \cdot (r,a,p) \in \mathcal{E}_1 \cup \mathcal{E}_2 \wedge p \Longrightarrow r) \\
&\qquad \wedge (\forall r, s \in \mathcal{C} \cdot p \Longrightarrow r \wedge s \Longrightarrow q \text{ implies } s = q)\} \\
\mathcal{K}(p \xrightarrow{a} q) &= \bigvee(\{\mathcal{K}_1(r \xrightarrow{a}_1 s) \mid r \xrightarrow{a}_1 s \wedge p \Longrightarrow r \wedge q \Longrightarrow s\} \\
&\qquad \cup \{\mathcal{K}_2(r \xrightarrow{a}_2 s) \mid r \xrightarrow{a}_2 s \wedge p \Longrightarrow r \wedge q \Longrightarrow s\}) \\
\mathcal{D}^{\mathcal{C}}(p) &= \bigcap\{\mathcal{D}_1(q) \cap \mathcal{D}_2(q) \mid q \in \mathcal{C} \wedge p \Longrightarrow q\}
\end{aligned}$$

(where R^* denotes the transitive closure of R). Note that, in the definition of $\mathcal{D}^{\mathcal{C}}$, we take $\mathcal{D}_i(p) = \mathcal{D}$ if $p \notin \mathcal{C}_i$.

We wish to show that $\mathcal{G} = \mathcal{G}_1 \sqcup \mathcal{G}_2$. We will omit the proof that, if $\mathcal{S}$ is anti-symmetric, then $\mathcal{G}$ is a weak schema, and $\mathcal{G}_1 \sqsubseteq \mathcal{G}$ and $\mathcal{G}_2 \sqsubseteq G$, and continue with the proof that $\mathcal{G}$ must be the least such weak schema.

Suppose $G' = (\mathcal{C}', \mathcal{E}', \mathcal{S}', \mathcal{K}', \mathcal{D}')$ is such that $\mathcal{G}_1 \sqsubseteq \mathcal{G}'$ and $\mathcal{G}_2 \sqsubseteq \mathcal{G}'$. Then $\mathcal{C}_1 \subseteq \mathcal{C}'$ and $\mathcal{C}_2 \subseteq \mathcal{C}'$, so $\mathcal{C} \subseteq \mathcal{C}'$. Simillarly it is easy to show that $\mathcal{S} \subseteq \mathcal{S}'$ and, for each $p \in \mathcal{C}$, $\mathcal{D}'(p) \subseteq \mathcal{D}^{\mathcal{C}}(p)$. Hence, since $\mathcal{S}'$ is anti-symmetric, so is $\mathcal{S}$, and $\mathcal{G}$ is indeed a weak schema.

Suppose that $p, q \in \mathcal{C}$ and $a \in \mathcal{L}$ are such that $p \xrightarrow{a}_{\mathcal{G}} q$. Then there is an $s \in \mathcal{C}$ such that either $s \xrightarrow{a}_1 q$ or $s \xrightarrow{a}_2 q$ and $p \Longrightarrow s$. Hence there is an $r \in \mathcal{C}'$ such that $s \xrightarrow{a}_{\mathcal{G}'} r$ and $r \Longrightarrow_{\mathcal{G}'} q$. Hence, since $p \Longrightarrow_{\mathcal{G}'} r$, there is a $t \in \mathcal{C}'$ such that $p \xrightarrow{a}_{\mathcal{G}'} t$ and $t \Longrightarrow_{\mathcal{G}'} q$.

Suppose $p, q \in \mathcal{C}$, $r \in \mathcal{C}'$ and $a \in \mathcal{L}$ are such that $p \xrightarrow{a}_{\mathcal{G}} q$, $p \xrightarrow{a}_{\mathcal{G}'} r$ and $r \Longrightarrow_{\mathcal{G}'} q$. Then for each $s, t \in \mathcal{C}_1$ such that $s \xrightarrow{a}_1 t$ and $p \Longrightarrow_{\mathcal{G}} s$ and $p \Longrightarrow_{\mathcal{G}} s$ we have $p \Longrightarrow_{\mathcal{G}'} s$ and $r \Longrightarrow_{\mathcal{G}'} t$ and so $\mathcal{K}_1(s \xrightarrow{a}_1 t) \leq \mathcal{K}'(p \xrightarrow{a}_{\mathcal{G}'} r)$. Similarly for $\mathcal{G}_2$. Hence $\mathcal{K}(p \xrightarrow{a}_{\mathcal{G}} q) \leq \mathcal{K}'(p \xrightarrow{a}_{\mathcal{G}'} r)$.

Hence we have all the conditions necessary for $\mathcal{G} \sqsubseteq \mathcal{G}'$. ∎

We say a finite collection of weak schemas, $\mathcal{G}_1, \ldots, \mathcal{G}_n$, is **compatible** if the relationship $(\mathcal{S}_1 \cup \ \ldots \ \cup \mathcal{S}_n)^*$ is anti-symmetric (where $\mathcal{S}_1, \ldots, \mathcal{S}_n$ are the specialisation relations of $\mathcal{G}_1, \ldots, \mathcal{G}_n$ respectively).

Consequently we have shown that, for any finite compatible collection of proper schemas, $\mathcal{G}_1, \ldots, \mathcal{G}_n$, there exists a weak schema $\mathcal{G} = \bigsqcup_{i=1,\ldots,n} \mathcal{G}_i$. What we must now do is find a way of constructing a proper schema from $\mathcal{G}$, if at all possible.

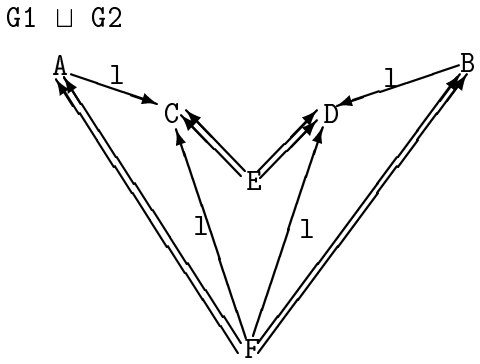


Figure 14: The least upper bound of two schemas

For example the two schemas, `G1` and `G2`, shown in figure 13 are compatible and their least upper bound is the weak schema shown in figure 14 (not showing arities).

Building proper schemas from weak schemas

In this section we will pay the price for our use of weak schemas. We need to find a way to introduce implicit classes into a weak schema $\mathcal{G}$, in order to form a schema $\overline{\mathcal{G}}$, such that if

there are any proper schemas greater than $\mathcal{G}$ then $\overline{\mathcal{G}}$ is such a schema.

First we introduce some new notations. For any $p \in \mathcal{C}$ and any $a \in \mathcal{L}$ we write $\mathcal{E}(p,a)$ to denote the set

$$\mathcal{E}(p,a) \;=\; \{q \in \mathcal{C} \mid p \xrightarrow{a} q\}$$

Further, for any set $X \subseteq \mathcal{C}$, we use $\mathcal{E}(X,a)$ to denote the set

$$\mathcal{E}(X,a) \;=\; \mathit{Min}_{\mathcal{S}}(\{q \in \mathcal{N} \mid \exists p \in X \cdot p \xrightarrow{a} q\})$$

where the function $\mathit{Min}_{\mathcal{S}} : \mathcal{P}(\mathcal{N}) \to \mathcal{P}(\mathcal{N})$ is defined so that, for any set $X \subseteq \mathcal{N}$, $\mathit{Min}_{\mathcal{S}}(X)$ is the set on minimal elements of X under the ordering $\mathcal{S}$. That is

$$\mathit{Min}_{\mathcal{S}}(X) \;=\; \{p \in X \mid \forall q \in X \cdot q \Longrightarrow p \text{ implies } q = p\}$$

We now proceed to build a weak schema $\overline{\mathcal{G}} = (\overline{\mathcal{C}}, \overline{\mathcal{E}}, \overline{\mathcal{S}}, \overline{\mathcal{K}}, \overline{\mathcal{D}})$ from $\mathcal{G}$ as follows:

1. First we will construct a set, $\mathit{Imp} \subseteq \mathcal{P}(\mathcal{C})$, of sets of classes, corresponding to our implicit classes. We will construct Imp via a series of auxiliary definitions as follows:

$$\begin{aligned} I^0 &= \{\{p\} \mid p \in \mathcal{C}\} \\ I^{n+1} &= \{\mathcal{E}(X,a) \mid X \in I^n, a \in \mathcal{L}\} \\ I^\infty &= \bigcup_{n=1}^{\infty} I^n \\ \mathit{Imp} &= \{X \in I^\infty \mid |X| > 1\} \end{aligned}$$

   So, intuitively, Imp is the set of all sets of classes with cardinality greater than 1 which one can reach by following a series of arrows from some class in $\mathcal{C}$. Note that, since there are only finitely many subsets of $\mathcal{N}$, we will eventually find an I_n such that $I_n \subseteq \bigcup_{i=1}^{n-1}(I_i)$, at which point we con stop computing further I_n's. Consequently the process of computing Imp will halt.

   Using the example in figure 14,

$$\begin{aligned} I^0 &= \{\{\mathtt{A}\},\{\mathtt{B}\},\{\mathcal{C}\},\{\mathtt{D}\},\{\mathtt{E}\},\{\mathtt{F}\}\} \\ I^1 &= \{\{\mathcal{C}\},\{\mathtt{D}\},\emptyset,\{\mathcal{C},\mathtt{D}\}\} \\ I^2 &= \{\emptyset\} \\ I^\infty &= I^0 \cup I^1 \\ \mathit{Imp} &= \{\mathit{Min}_{\mathcal{S}}(\{\mathcal{C},\mathtt{D}\})\} = \{\{\mathcal{C},\mathtt{D}\}\} \end{aligned}$$

2. We define $\overline{\mathcal{C}}$ by first taking $\mathcal{C}$ and then adding a new class $\overline{X}$ for every $X \in Imp$. That is

$$\overline{\mathcal{C}} = \mathcal{C} \cup \{\overline{X} \mid X \in Imp\}$$

Continuing with figure 14,

$$\overline{\mathcal{C}} = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}, \overline{\{\mathtt{C}, \mathtt{D}\}}\}$$

3. We define $\overline{\mathcal{E}}$ so that if $p \xrightarrow{a} q$ for each $q \in X$ then $(p, a, \overline{X}) \in \overline{\mathcal{E}}$, while if there is a unique q such that $p \xrightarrow{a} q$ then $(p, a, q) \in \overline{\mathcal{E}}$. Formally:

$$\begin{aligned}\overline{\mathcal{E}} &= \{(x, a, q) \mid x \in \overline{\mathcal{C}}, a \in \mathcal{L}, \mathcal{E}(x, a) = \{q\}\} \\ &\quad \cup \{(x, a, \overline{Y}) \mid x \in \overline{\mathcal{C}}, a \in \mathcal{L}, \mathcal{E}(x, a) = Y \wedge |Y| > 1\}\end{aligned}$$

where $\mathcal{E}(\overline{X}, a) = \mathcal{E}(X, a)$ for all $X \in Imp$.

For our running example,

$$\overline{\mathcal{E}} = \{(\mathtt{A}, \mathtt{l}, \mathtt{C}), (\mathtt{B}, \mathtt{l}, \mathtt{D}), (\mathtt{F}, \mathtt{l}, \mathtt{C}), (\mathtt{F}, \mathtt{l}, \mathtt{D}), (\mathtt{F}, \mathtt{l}, \overline{\{\mathtt{C}, \mathtt{D}\}})\}$$

4. We define the relation $\overline{\in}$ between $\mathcal{C}$ and $\overline{\mathcal{C}}$ so that, for any $p \in \mathcal{C}$ and $x \in \overline{\mathcal{C}}$, we have $p \overline{\in} x$ iff either $x \equiv p$ or $x = \overline{X}$ for some $X \in Imp$ and $p \in X$.

We define $\overline{\mathcal{S}}$ by first taking $\mathcal{S}$ and then adding every pair $(\overline{X}, \overline{Y})$ such that every class in Y has a specialisation in X; every pair $(\overline{X}, p)$ where p has a specialisation in X; and every pair $(p, \overline{X})$ where p is a specialisation of every class in X.

$$\overline{\mathcal{S}} = \{(x, y) \mid x, y \in \overline{\mathcal{C}},\ \forall q \overline{\in} y \cdot \exists p \overline{\in} x \cdot (p, q) \in \mathcal{S}\}$$

Note that $\mathcal{S} \subseteq \overline{\mathcal{S}}$.

Thus, for figure 14,

$$\overline{\mathcal{S}} = \mathcal{S} \cup \{\ \overline{\{\mathtt{C}, \mathtt{D}\}} \Longrightarrow \overline{\{\mathtt{C}, \mathtt{D}\}}\ \} \cup \{\ \overline{\{\mathtt{C}, \mathtt{D}\}} \Longrightarrow \mathtt{C},\ \overline{\{\mathtt{C}, \mathtt{D}\}} \Longrightarrow \mathtt{D}\ \} \cup \{\mathtt{E} \Longrightarrow \overline{\{\mathtt{C}, \mathtt{D}\}}\ \}$$

5. $\overline{\mathcal{K}}$ is defined so that any edge $x \xrightarrow[\overline{\mathcal{G}}]{a} y$ in $\overline{\mathcal{G}}$ has a arity which is the least upper bound of the arities of the corresponding edges in $\mathcal{G}$.

$$\overline{\mathcal{K}}(x \xrightarrow[\overline{\mathcal{G}}]{a} y) = \bigvee \{\mathcal{K}(p \xrightarrow[\mathcal{G}]{a} q) \mid p, q \in \mathcal{C}, p \overline{\in} x \wedge q \overline{\in} y \wedge p \xrightarrow[\mathcal{G}]{a} q\}$$

6. We define $\overline{\mathcal{D}}$ by extending the definition of $\mathcal{D}^{\mathcal{C}}$ to $\overline{\mathcal{C}}$ so that $\overline{\mathcal{D}}(\overline{X})$ is the intersection of the sets $\mathcal{D}^{\mathcal{C}}(p)$ for $p \in X$.

$$\begin{aligned}\overline{\mathcal{D}}(p) &= \mathcal{D}^{\mathcal{C}}(p) && \text{for } p \in \mathcal{C} \\ \overline{\mathcal{D}}(\overline{X}) &= \textstyle\bigcap_{p \in X} \mathcal{D}^{\mathcal{C}}(p) && \text{for } X \in Imp\end{aligned}$$

Summarizing the effect on figure 14, we get the schema G3 in figure 13, with the class ? replaced by the class $\overline{\{\mathtt{C},\mathtt{D}\}}$.

Lemma 5.3 *For any $X \in Imp$, $(p, q \in X \wedge p \Longrightarrow q)$ implies $(p = q)$.*

Proof: Imediate from the definition of Imp. ∎

Lemma 5.4 *If $\mathcal{S}$is anti-symmetric then so is $\overline{\mathcal{S}}$.*

Proof: Suppose $\overline{\mathcal{S}}$ is not anti-symmetric. Then there exist $x, y \in \overline{\mathcal{C}}$ such that $x \neq y$, $(x, y) \in \overline{\mathcal{S}}$ and $(y, x) \in \overline{\mathcal{S}}$. We must show that $\mathcal{S}$ is not anti-symmetric.

Case 1: $x = p \in \mathcal{C}$ and $y = q \in \mathcal{C}$. Then $(p, q) \in \mathcal{S}$ and $(q, p) \in \mathcal{S}$. Hence, since $p \neq q$, $\mathcal{S}$ is not anti-symmetric.

Case 2: $x = p \in \mathcal{C}$ and $y = \overline{Y}$ where $Y \in Imp$. Then, since $(\overline{Y}, p) \in \overline{\mathcal{S}}$, there is a $q \in Y$ such that $q \Longrightarrow p$. But, since $(p, \overline{Y}) \in \overline{\mathcal{S}}$, $p \Longrightarrow q$. Suppose $p = q$, so that $p \in Y$. Then, since for every $r \in Y$, $p \Longrightarrow r$, it follows from Lemma 5.3 that $Y = \{p\}$ (a contradiction). Hence $p \neq q$ and $\mathcal{S}$ is not anti-symmetric.

Case 3: $x = \overline{X}$ and $y = \overline{Y}$ where $X, Y \in Imp$. Suppose $p \in X$. Then, since $(\overline{Y}, \overline{X}) \in \overline{\mathcal{S}}$, there is a $q \in Y$ such that $q \Longrightarrow p$. But similarly there is a $r \in X$ such that $r \Longrightarrow q$. Hence we have $r \Longrightarrow p$, and from Lemma 5.3 we get $r = p$ and $p \Longrightarrow q \Longrightarrow p$. If $\mathcal{S}$ were anti-symmetric then this would imply that $p = q$, and consequently we would get that, for any $p \in \mathcal{C}$, $p \in X$ iff $p \in Y$. This contradicts the fact that $X \neq Y$ and so $\mathcal{S}$ is not anti-symmetric. ∎

Proposition 5.5

1. *For any $x, y, z \in \overline{\mathcal{C}}$ and any $a \in \mathcal{L}$, if $(x, a, y) \in \overline{\mathcal{E}}$ and $(x, a, z) \in \overline{\mathcal{E}}$ then $y = z$.*
2. *$\overline{\mathcal{G}}$ is a weak schema.*

Proof:

1. This follows immediately from the definition of $\overline{\mathcal{G}}$.
2. We will omit the proofs that $\overline{\mathcal{S}}$ is a transitive and reflexive (reflexivity is immediate, and transitivity, while involving some tedious case analysis, is also straight forward), and that, if $(x, y) \in \overline{\mathcal{S}}$ then $\overline{\mathcal{D}}(y) \subseteq \overline{\mathcal{D}}(x)$. We shall however show that, if $(x, a, u) \in \overline{\mathcal{E}}$ and $(y, x) \in \overline{\mathcal{S}}$ then there is a $v \in \overline{\mathcal{C}}$ such that $(y, a, v) \in \overline{\mathcal{E}}$ and $(v, u) \in \overline{\mathcal{S}}$.

Case 1: $p, q \in \mathcal{C}$ are such that $(p, q) \in \overline{\mathcal{S}}$ and $(q, a, x) \in \overline{\mathcal{E}}$. Then $p \Longrightarrow q$, hence for all $r \in \mathcal{E}(q, a)$ there exists an $s \in \mathcal{E}(p, a)$ such that $s \Longrightarrow r$. Hence there exists a $y \in \overline{\mathcal{C}}$ such that $(p, a, y) \in \overline{\mathcal{E}}$ and $(y, x) \in \overline{\mathcal{S}}$.

Case 2: $p \in \mathcal{C}$, $X \in Imp$ are such that $(p, \overline{X}) \in \overline{\mathcal{S}}$ and $(\overline{X}, a, y) \in \overline{\mathcal{E}}$. Then, for every $q \in X$, $p \Longrightarrow q$, so for every $r \in \mathcal{E}(X, a)$ there is an $s \in \mathcal{E}(p, a)$ such that $s \Longrightarrow r$. Hence there is a $x \in \overline{\mathcal{C}}$ such that $(p, a, x) \in \overline{\mathcal{E}}$ and $(x, y) \in \overline{\mathcal{S}}$.

The other two cases follow along similar lines.

■

Lemma 5.6 *For any weak schema $\mathcal{G}$, $\mathcal{G} \sqsubseteq \overline{\mathcal{G}}$.*

Proof: Straight forward from definition of $\overline{\mathcal{G}}$. ■

We say that a weak schema, $\mathcal{G} = (\mathcal{C}, \mathcal{E}, \mathcal{S}, \mathcal{D}^{\mathcal{C}})$ is **consistent** iff for every $x \in \overline{\mathcal{C}}$, $\overline{\mathcal{D}}(x) \neq \emptyset$. We say that $\mathcal{G}$ is **inconsistent** otherwise. It would be possible to have a more sophisticated idea of consistency by introducing an special binary "consistency relationship" on $\mathcal{C}$, and requiring that, for every $X \in Imp$ and every $p, q \in X$, the pair (p, q) is in the consistency relationship. However this isn't terribly interesting so we won't pursue it any further here.

From Proposition 5.5 it follows that $\overline{\mathcal{G}}$ is a proper schema iff $\mathcal{G}$ is consistent.

Proposition 5.7 *For any weak schema $\mathcal{G}$, $\mathcal{G}$ is consistent iff there is a proper schema $\mathcal{G}'$ such that $\mathcal{G} \sqsubseteq \mathcal{G}'$.*

Proof: ($\Longrightarrow$): If $\mathcal{G}$ is consistent then $\overline{\mathcal{G}}$ is such a proper schema.

($\Longleftarrow$): Suppose that $\mathcal{G}' = (\mathcal{C}', \mathcal{E}', \mathcal{S}', \mathcal{D}')$ is a weak schema such that $\mathcal{G} \sqsubseteq \mathcal{G}'$, and that $\mathcal{G}$ is inconsistent.

Suppose $\overline{\mathcal{D}}(x) = \emptyset$ for some $x \in \overline{\mathcal{C}}$. If $x = p \in \mathcal{C}$ then $\mathcal{D}'(p) \subseteq \mathcal{D}^{\mathcal{C}}(p) = \emptyset$ and so $\mathcal{G}'$ is not a proper schema. Suppose $x = \overline{X}$ where $X \in Imp$. Then there is a class $p \in \mathcal{C}$ and a series of arrows $a_1, \ldots, a_n \in \mathcal{L}$ such that, for every $q \in X$

$$p \xrightarrow{a_1} \ldots \xrightarrow{a_n} q$$

We can show by induction on n that, for every $q \in X$ there is an $r \in \mathcal{C}'$ such that

$$p \xrightarrow{a_1}{}' \ldots \xrightarrow{a_n}{}' r$$

and $r \Longrightarrow' q$. But if $\mathcal{G}'$ is a proper schema then there can be at most one such r. Hence, for every $q \in X$, $r \Longrightarrow' q$, and so $\mathcal{D}'(r) \subseteq \mathcal{D}'(q) \subseteq \mathcal{D}^{\mathcal{C}}(q)$. Hence, since $\overline{\mathcal{D}}(\overline{X}) = \emptyset$, we get $\mathcal{D}'(r) = \emptyset$, and $\mathcal{G}'$ is not a proper schema. ∎

We would like to show that, if $\mathcal{G}$ is consistent, then $\overline{\mathcal{G}}$ is the least proper schema greater than $\mathcal{G}$. Unfortunately this isn't quite true. Firstly it is possible to form other similar proper schemas by using different names for the implicit classes (compare this to alpha-conversion in the lambda calculus). Secondly, for any two sets $X, Y \in Imp$, if every class in Y has a specialisation in X then our method will include the pair $(\overline{X}, \overline{Y})$ in $\overline{\mathcal{S}}$. However it is not necessarily the case that this specialisation relation is necessary, and it might be safe to omit it. We could attempt to modify our method so that such pairs are only introduced when necessary. Instead we will argue that, since the implicit classes have no additional information associated with them, it follows that these specialisation relations do not introduce any extra information into the database schema, and so, since they seem natural, it is best to leave them there. Consequently we feel justified in making the following definitions:

A finite collection of proper database schemas, $\mathcal{G}_1, \ldots, \mathcal{G}_n$, is said to be **consistent** if they are compatible and the weak schema $\mathcal{G} = \bigsqcup_{i=1}^{n} \mathcal{G}_i$ is consistent.

Given a consistent collection of database schemas, $\mathcal{G}_1, \ldots, \mathcal{G}_n$, the **merge** of $\mathcal{G}_1, \ldots, \mathcal{G}_n$ is the database schema $\overline{\mathcal{G}}$, where $\mathcal{G} = \bigsqcup_{i=1}^{n} \mathcal{G}_i$.

5.3 Incorporating Correspondences Between Classes

In Section 5.1 we stated that one of the first problems involved in schema merging is identifying the correspondences between classes of the different database schemas. However we went on to say that these correspondences must be identified by the user, and that he must alter the database schemas accordingly before commencing the computation of their merge. However, in that section, we limited ourselves to considering only exact correspondences of classes, that is, where two classes represent the same class of real world objects. In this section we will show how to alter database schemas, prior to merging, in order to reflect some slightly more general correspondences.

For any two schemas, $\mathcal{G}_1$ and $\mathcal{G}_2$, and any classes p and q, of $\mathcal{G}_1$ and $\mathcal{G}_2$ respectively, we will allow the user to make one of the following three kinds of assertions relating p to q:

1. $p = q$: meaning that p and q represent the same real world classes of objects.
2. $p \subseteq q$: meaning that every object of class p is also an object of class q.
3. $p \supseteq q$: meaning every object of class q is also an object of class p.

Alternatively the user is not actually obliged to state any correspondence involving p and q. It should be noted that we are considering the real-world or conceptual classes represented by the abstract database classes here: the correspondences between the objects represented in the databases are another matter. In other words, just because we assert a correspondence $p \subseteq q$ this does not necessarily mean that we expect that for every object of class p stored in the first database there will be a corresponding object of class q stored in the second database.

Suppose we start with a collection of database schemas, $\mathcal{G}_1, \ldots, \mathcal{G}_n$, which are to be merged. We will assume that initially the databases have disjoint sets of classes, and that we have a set of correspondences between classes $\{p_1 \# q_1, \ldots, p_k \# q_k\}$ where each $\#$ is one of $=$, $\subseteq$ or $\supseteq$, and for any pair of classes there can be at most one correspondence between them in the set.

We will add a new, initially empty schema, $\mathcal{G}^C$, to the set of schemas to be merged. We call $\mathcal{G}^C$ the *correspondence schema* and use it to encode the $\subseteq$ and $\supseteq$ correspondences between classes.

For each pair of schemas, $\mathcal{G}_i$ and $\mathcal{G}_j$, and each pair of classes, p and q from $\mathcal{G}_i$ and $\mathcal{G}_j$ respectively, we carry out the following modifications to the schemas:

1. If we have a correspondence $p = q$ then replace all occurrences of q in all the schemas by p. (It is not sufficient just to replace occurences of q in $\mathcal{G}_j$ since q may have been introduced into some of the other schemas).

2. If $p \subseteq q$ then add the classes p and q to $\mathcal{G}^C$ (with the $\mathcal{D}$ as their printable value sets), and then extend its specialisation relation so that it contains the relation $p \Longrightarrow q$ (and it remains transitive and reflexive).

3. If $p \supseteq q$ then add the classes p and q to $\mathcal{G}^C$, and then extend its specialisation relation so that it contains the relation $q \Longrightarrow p$.

4. If there is no correlation between p and q then leave the schemas unchanged.

Once we have carried out the above modifications, if $\mathcal{G}_1, \ldots, \mathcal{G}_n$ and $\mathcal{G}^C$ are still schemas (that is, their specialisation relationships remain anti-symmetric), then we can proceed to try to find their merge, as described in Section 5.1. If one or more of them has acquired a non-antisymmetric specialisation relation then we reject the set of correspondences claiming it is inconsistent.

5.4 Meta-schemas

We demonstrated in section 2.1 that ER schemas can be embedded in our model in a natural way. We then went on in sections 5.1 and 5.2 to find a way of determining the merge of a collection of schemas whenever it exists. In order for this conversion-and-merging process to be useful for ER schemas, we must show that the schema resulting from such a merge can also be considered to be an embedding of an ER schema. We could then reverse the conversion process in order to find an ER schemas which is the merge of our original collection of ER schemas.

Compared to our model, the most significant restriction in the ER model is that classes are stratified into three disjoint sets: entity sets, relationships and base types. Furthermore, specialization relationships are constrained so that they may only occur between classes in the same stratum, and the arrows of a class belonging to one stratum must have classes in the next stratum.

In this section we will look at the problem of imposing such constraints on schemas, and will show that these constraints are, in some sense, preserved by our merging process. While ER-like models always divide classes up into three distinct strata and only allow arrows to go from one stratum to the next, we will present a more general form of constraint, which we will call a *meta-schema*, which divides the classes into any number of distinct sets and then imposes restrictions on which arrows may go from classes in each set to classes in each other set.

A **meta-schema** over classes $\mathcal{N}$ and arrow labels $\mathcal{L}$ consists of a pairwise disjoint collection of sets of classes, $\{\mathcal{N}_i \mid i \in I\}$, such that $\mathcal{N} = \bigcup_{i \in I} \mathcal{N}_i$, and a collection of sets of labels $\{\mathcal{L}_{i,j} \mid i, j \in I\}$ such that, for each $i \in I$, the collection of sets of labels $\{\mathcal{L}_{i,j} \mid j \in I\}$ is pairwise disjoint.

A schema $\mathcal{G} = (\mathcal{C}, \mathcal{E}, \mathcal{S}, \mathcal{D}^{\mathcal{C}})$ is said to **satisfy** a meta-schema $(\{\mathcal{N}_i \mid i \in I\}, \{\mathcal{L}_{i,j} \mid i, j \in I\})$ iff

1. If $p \Longrightarrow q \in \mathcal{S}$ then $p \in \mathcal{N}_i$ and $q \in \mathcal{N}_i$ for some $i \in I$.
2. If $p \xrightarrow{a} q \in \mathcal{E}$ then $p \in \mathcal{N}_i$, $q \in \mathcal{N}_j$ and $a \in \mathcal{L}_{i,j}$ for some $i, j \in I$.

for all $a \in \mathcal{L}$ and $p, q \in \mathcal{C}$.

So the classes are divided into the sets $\{\mathcal{N}_i\}$, and two classes can only be related by a specialization relation if they belong to the same set, and, in addition, any class in a set $\mathcal{N}_i$ can only have an a-arrow with a class in the set $\mathcal{N}_j$ if a is in the set of labels $\mathcal{L}_{i,j}$.

For example, the schema shown in figure 2 satisfies the meta-schema given by taking $I = \{0, 1, 2\}$ and

$$\begin{aligned} \mathcal{N}_0 &= \{\texttt{int, str}\} \\ \mathcal{N}_1 &= \{\texttt{Dog, Person}\} \\ \mathcal{N}_2 &= \{\texttt{Owned-by}\} \\ \mathcal{L}_{1,0} &= \{\texttt{age, name}\} \\ \mathcal{L}_{2,1} &= \{\texttt{dog, owner}\} \end{aligned}$$

and $\mathcal{L}_{i,j} = \emptyset$ for all other $i, j \in I$. In general any ER-structure will translate to a schema satisfying a meta-schema of the form found in figure 15.

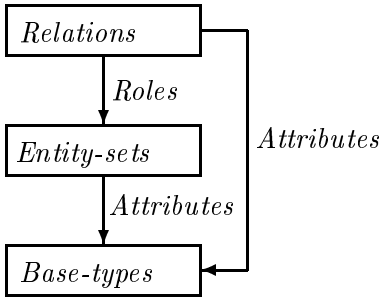


Figure 15: A meta-schema for ER-structures

Note that our definition of *satisfaction* of a meta-schema by a schema applies equally well to weak schemas and proper schemas.

Proposition 5.8 *If $\mathcal{G}_1$ and $\mathcal{G}_2$ are compatible weak schemas satisfying the meta-schema $\mathcal{M}$, then $\mathcal{G}_1 \sqcup \mathcal{G}_2$ also satisfies $\mathcal{M}$.*

Proof: Let $\mathcal{G} = \mathcal{G}_1 \sqcup \mathcal{G}_2$ as constructed in proposition 5.2. To prove the first condition in the definition of satisfaction first note that, for every pair of classes $(p, q) \in \mathcal{S}_1 \cup \mathcal{S}_2$, either $(p, q) \in \mathcal{S}_1$ or $(p, q) \in \mathcal{S}_2$, so there is an $i \in I$ such that $p \in \mathcal{N}_i$ and $q \in \mathcal{N}_i$. Clearly then, the condition holds for the reflexive closure of $\mathcal{S}_1 \cup \mathcal{S}_2$. We can prove the condition for $(\mathcal{S}_1 \cup \mathcal{S}_2)^*$, that is the transitive reflexive closure of $\mathcal{S}_1 \cup \mathcal{S}_2$, by induction on the number of steps in forming the transitive closure.

The second condition for satisfaction of $\mathcal{M}$ follows from the observation that $\mathcal{E} \subseteq \mathcal{E}_1 \cup \mathcal{E}_2$. ■

Suppose we have a consistent weak schema, $\mathcal{G}$, satisfying the meta-schema $\mathcal{M}$: we would like to show that the proper schema $\overline{\mathcal{G}}$ also satisfies $\mathcal{M}$. Unfortunately this is not quite true since $\overline{\mathcal{G}}$ may involve additional classes not in $\mathcal{N}$. However we can prove the following results:

Lemma 5.9 *If $Imp \subseteq \mathcal{P}(\mathcal{N})$ is constructed from $\mathcal{G}$ as in section 5.2, then, for each $X \in Imp$, there is an i such that $X \subseteq \mathcal{N}_i$.*

Proof: From the construction of *Imp* it is sufficient to prove that, for each set I_k, and each $X \in I_k$ there is an i such that $X \subseteq \mathcal{N}_i$. This follows by induction on k. ■

Proposition 5.10 *Suppose $\mathcal{G}$ is a consistent weak schema satisfying $\mathcal{M}$. Then the proper schema $\overline{\mathcal{G}}$ satisfies the meta-schema $\overline{\mathcal{M}} = (\{\overline{\mathcal{N}}_i \mid i \in I\}, \{\mathcal{L}_{i,j} \mid i, j \in I\})$ where*

$$\overline{\mathcal{N}}_i = \mathcal{N}_i \cup \{\overline{X} \mid X \subseteq \mathcal{N}_i\}$$

for $i \in I$.

Proof: This follows from from lemma 5.9 and inspection of the construction of $\overline{\mathcal{G}}$ in section 5.2. In particular it is enough to observe that the specialisation relations added to $\overline{\mathcal{S}}$ and the arrows added to $\overline{\mathcal{E}}$ do not violate either of the conditions necessary for satisfaction of $\overline{\mathcal{M}}$. ■

Hence we can deduce the following theorem which states that. in some sense, satisfaction of meta-schemas is preserved by our merging process:

Theorem 5.11 *If $\mathcal{M} = (\{\mathcal{N}_i \mid i \in I\}, \{\mathcal{L}_{i,j} \mid i, j \in I\})$ is a meta-schema over $\mathcal{N}$ and $\mathcal{L}$, and $\mathcal{G}_1, \ldots, \mathcal{G}_n$ is a consistant collection of proper schemas satisfying $\mathcal{M}$ such that $\overline{\mathcal{G}}$ is the merge of $\mathcal{G}_1, \ldots, \mathcal{G}_n$, then $\overline{\mathcal{G}}$ satisfies the meta-schema $\overline{\mathcal{M}}$.*

Proof: Follows directly from proposition 5.8 and proposition 5.10. ■

It follows that if we use meta-schemas to constrain the schemas we are merging we can expect similar constraints to apply to the merge of those schemas. We can use this principle to ensure other restrictions as well. For example, if we wished to ensure that no specialization relationships were inserted between base types and no new classes were introduced below any pair of base types in the schema shown in figure 2, we could do so by splitting the set $\mathcal{N}_0$ into the two unary sets {`int`} and {`str`} (with the appropriate changes to I and the sets $\mathcal{L}_{i,j}$).

6 Lower Merges

In Section 5.2 we defined the merge of a collection of schemas as their *least upper bound* under an information ordering. A consequence of this is that, if we merge a number of schemas, then any instance of the merged schema can be considered to be an instance of any of the schemas being merged. In some cases, however, it is desirable to find the *greatest lower bound* of a collection of schemas and use that as the merge. In this case any instances of the schemas being merged would also be instances of the merged schema, and, further, we would expect to be able to coalesce or take the union of a number of instances of the collection of schemas and use that as an instance of the merged schema. This kind of merge is likely to arise in, for example, the formulation of federated database systems.

We will refer to the merges defined in section 5.2 as **upper merges**, and, in this section, we will discuss the formulation of **lower merges**, representing the greatest lower bound of a collection of schemas. It could be legitimately argued that lower merges are of primary importance and should have been introduced first. However we introduced upper merges as our primary concept of a merge because they are inherently simpler and more natural to formulate. There are a number of complications involved in giving a formal definition of lower merges.

As it stands, taking the lower bound of a collection of schemas using our established information ordering is clearly unsatisfactory: any information on which two schemas disagree on is lost. For example if one schema has the class *Dog* with arrows *name* and *age*, and another has *Dog* with arrows *name* and *breed*, then in the lower bound of the two schemas the class *Dog* will only have the arrow *name*. What we want, however, is some way of saying that instances of the class *Dog may* have *age*-arrows and *may* have *breed*-arrows, but are not necessarily required to have either. Worse still, if one schema has the class *Guide-Dog* and another does not, then the lower bound of the two schemas will not. This second problem can be dealt with by choosing a slightly different ordering on schemas, requiring that for one schema to be bellow another in the ordering it must have all the classes of the second schema. In some sense we could argue that such an ordering would be more correct, as an information ordering, than the one defined in section 5.2, if we considered a smaller set of classes to put more restrictions on the classes to which the objects of an instance can belong. However the first problem, that of losing arrows, is more difficult to deal with, and will require us first to extend our definition of arity constraints.

6.1 Extended arity constraints

We will extend the lattice of arity constraints defined in section 2.1 with the additional constraint 0-0, forming the semi-lattice shown in figure 16. The idea here is that if a class

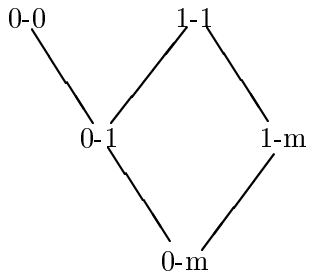


Figure 16: New semi-lattice of arity constraints

p had an a-arrow with arity constraint 0-0 then no object of class p can have any a-arrows. However we will not allow arrows with arity constraints of 0-0 in our schemas: instead we will assume that any class p which has no a-arrows behaves as if it had an implicit a-arrow with arity constraint 0-0.

Now, if one schema has an arrow which is not included in another, then we can pretend that the second schema also has the arrow, but with arity constraint 0-0, and we can take the greatest lower bound of the arity constraints (under the ordering $\leq$) to be the arity constraint of the arrow in the merged schema.

6.2 Lower weak schemas

In our construction of lower merges we will need to introduce implicit classes similar to those used in section 5.2, except that an implicit class now represents an object having a choice of belonging to any of a number of proper classes, rather than that an object must belong to each of a number of proper classes. Consequently we must change our definition of *weak schemas* to reflect the fact that a class having multiple a-arrows represents that each instance of the class may have a-arrows going to objects of any one of a number of classes, rather than that each a-arrow of an object of the first class must go to an object belonging to all of the later classes.

A **lower weak schema** over class names $\mathcal{N}$ and arrow labels $\mathcal{L}$ is a four-tuple $(\mathcal{C}, \mathcal{E}, \mathcal{S}, \mathcal{K})$, where $\mathcal{C} \subseteq \mathcal{N}$, $\mathcal{E} \subseteq \mathcal{C} \times \mathcal{L} \times \mathcal{C}$, $\mathcal{S} \subseteq \mathcal{C} \times \mathcal{C}$ and $\mathcal{K} : \mathcal{E} \to \{0\text{-m}, 1\text{-m}, 0\text{-}1, 1\text{-}1\}$, such that

LW1 If $p, q, r \in \mathcal{C}$ and $a \in \mathcal{L}$ are such that $p \xrightarrow{a} q$ and $p \xrightarrow{a} r$ and $q \Longrightarrow r$ then $q = r$.

LW2 $\mathcal{S}$ is a partial order on $\mathcal{C}$.

LW3 For any $p, q, r \in \mathcal{C}$ and $a \in \mathcal{L}$ if $p \Longrightarrow q$ and $p \xrightarrow{a} r$ and there is an s such that $q \xrightarrow{a} s$ then there exists a $t \in \mathcal{C}$ such that $q \xrightarrow{a} t$ and $r \Longrightarrow t$.

LW4 For any $p, q, r \in \mathcal{C}$ and any $a \in \mathcal{L}$, if $p \xrightarrow{a} q$ and $p \xrightarrow{a} r$ then $\mathcal{K}(p \xrightarrow{a} q) = \mathcal{K}(p \xrightarrow{a} r)$.

LW5 For any $p, q, r \in \mathcal{C}$ and any $a \in \mathcal{L}$, if $p \Longrightarrow q$ and $q \xrightarrow{a} r$ then there exists an $s \in \mathcal{C}$ such that $p \xrightarrow{a} s$ and $\mathcal{K}(p \xrightarrow{a} s) \leq \mathcal{K}(q \xrightarrow{a} r)$.

The main difference between lower weak schemas and the weak schemas defined in section 5.2 is in the axiom **LW3**. This axiom ensures that the axiom **A2** holds when we convert our lower weak schemas to proper schemas by replacing multiple a-arrows with single a-arrows to implicit classes.

We will not take printable value sets into account in our study of lower merges though the necessary extensions are straight forward.

6.3 Another information ordering and lower merges

We will define our lower merges to be greatest lower bounds under an information ordering on lower weak schemas. However, for reasons already discussed, we will not use the same ordering as in section 5.2, but instead will define a new ordering which preserves more of the information of the schemas being merged.

We define the ordering $\sqsubseteq^L$ on lower weak schemas such that, if $\mathcal{G}_1$ and $\mathcal{G}_2$ are l-w-schemas over $\mathcal{N}$ and $\mathcal{L}$, then $\mathcal{G}_1 \sqsubseteq^L \mathcal{G}_2$ iff

1. $\mathcal{C}_1 \supseteq \mathcal{C}_2$.
2. If $p \Longrightarrow_1 q$ and $p \in \mathcal{C}_2$ or $q \in \mathcal{C}_2$ then $p \Longrightarrow_2 q$.
3. If $p \xrightarrow{a}_2 q$ then there exists an $r \in \mathcal{C}_1$ such that $p \xrightarrow{a}_1 r$ and $q \Longrightarrow_1 r$.
4. For any $p \in \mathcal{C}_2$ and any $a \in \mathcal{L}$

$$\mathcal{K}'_1(p, a) \leq \mathcal{K}'_2(p, a)$$

where

$$\mathcal{K}'_i(p,a) \;=\; \begin{cases} \mathcal{K}_i(p \xrightarrow{a} q) & : \text{where } p \xrightarrow{a}_i q \\ & \text{if such a } q \text{ exists} \\ \text{0-0} & : \text{otherwise} \end{cases}$$

(We will use the $\mathcal{K}'$ notation again later on.)

The second condition here is perhaps sufficiently different from the conditions of the previous ordering to require some additional explanation: it means that, for one lower weak schema to contain less information than another, it must not assert any specialisation relations about the classes of the second schema that are not also asserted by the second schema, though it may assert any specialisation relations not involving the classes of the second schema.

Proposition 6.1 *$\sqsubseteq^L$ is a reflexive, anti-symmetric relation.*

Proof: Similar to that of proposition 5.1. ∎

Unfortunately $\sqsubseteq^L$ turns out not to be a transitive relation. A consequence of this is that, if we construct a binary merging operator, it fails to be associative. The best we can do is construct a binary lower merge operator and give a characterisation of those sets of lower weak schemas for which the operator will be associative and can be repeatedly applied to form a merge of the set. On the other hand one simplification of lower merges over upper merges is that we don't have to worry about compatibility of schemas: the greatest lower bound of a pair of schemas always exists.

Proposition 6.2 *For any pair of lower weak schemas, $\mathcal{G}_1$ and $\mathcal{G}_2$, their greatest lower bound under the ordering $\sqsubseteq^L$ is computable.*

Proof: Given any two lower weak schemas, $\mathcal{G}_1 = (\mathcal{C}_1, \mathcal{E}_1, \mathcal{S}_1, \mathcal{K}_1)$ and $\mathcal{G}_2 = (\mathcal{C}_2, \mathcal{E}_2, \mathcal{S}_2, \mathcal{K}_2)$, we define $\mathcal{G}$ by:

$$\begin{aligned}
\mathcal{C} &= \mathcal{C}_1 \cup \mathcal{C}_2 \\
\mathcal{S} &= \{(p,q) \in \mathcal{S}_1 \cup \mathcal{S}_2 \mid ((p,q) \in \mathcal{S}_1 \cap \mathcal{S}_2) \vee \\
&\qquad (p \notin \mathcal{C}_1 \wedge q \notin \mathcal{C}_1) \vee (p \notin \mathcal{C}_2 \wedge q \notin \mathcal{C}_2)\} \\
\mathcal{E} &= \{(p,a,q) \mid (\exists s \in \mathcal{C} \cdot (p,a,s) \in \mathcal{E}_1 \cup \mathcal{E}_2) \wedge \\
&\qquad (\exists r \in \mathcal{C} \cdot r \Longrightarrow p \wedge (r,a,q) \in \mathcal{E}_1 \cup \mathcal{E}_2) \wedge \\
&\qquad (\forall r, s \in \mathcal{C} \cdot r \Longrightarrow p \wedge q \Longrightarrow s \wedge (r,a,s) \in \mathcal{E}_1 \cup \mathcal{E}_2 \text{ implies } q \equiv s)\}
\end{aligned}$$

and, for any $p, q \in \mathcal{C}$ and $a \in \mathcal{L}$ such that $p \xrightarrow{a} q$,

$$\mathcal{K}(p \xrightarrow{a} q) = \begin{cases} \bigwedge\{\mathcal{K}'_1(r,a) \bigwedge \mathcal{K}'_2(r,a) \mid \\ \quad r \in \mathcal{C} \cdot p \Longrightarrow r \wedge \exists s \in \mathcal{C} \cdot r \xrightarrow{a} s\} \\ \qquad\qquad \text{:if } p \in \mathcal{C}_1 \cap \mathcal{C}_2 \\ \mathcal{K}'_1(p,a) \qquad \text{:if } p \in \mathcal{C}_1 \setminus \mathcal{C}_2 \\ \mathcal{K}'_2(p,a) \qquad \text{:if } p \in \mathcal{C}_2 \setminus \mathcal{C}_1 \end{cases}$$

where $\mathcal{K}'_i$ has the same meaning as before.

The proof that $\mathcal{G} = \mathcal{G}_1 \sqcap^L \mathcal{G}_2$ has three parts: showing that $\mathcal{G}$ is indeed a lower weak schema; showing that $\mathcal{G} \sqsubseteq^L \mathcal{G}_1$ and $\mathcal{G} \sqsubseteq^L \mathcal{G}_2$; and showing that, for any $\mathcal{G}_3$ such that $\mathcal{G}_3 \sqsubseteq^L \mathcal{G}_1$ and $G_3 \sqsubseteq^L \mathcal{G}_2$, we have $G_3 \sqsubseteq^L \mathcal{G}$. These proofs are fairly straight-forward and are similar to the proofs in proposition 5.2. We will not give details here. ∎

We must now find a way of catagorising those collections of schemas for which the greatest lower bound operator is associative. It might be possible to construct weaker restrictions than the following if we involved the arrow relations. It remains to be seen how major our restriction is in practice: that is, how likely it is that a set of schemas we wish to merge will violate the restriction.

A collection of lower weak schemas, $\mathcal{G}_1, \ldots, \mathcal{G}_n$, is said to be **well-behaved** iff, for any $\mathcal{G}_i$ and $\mathcal{G}_j$ and any classes p and q, if $p \Longrightarrow_i q$ and $p \Longrightarrow_j q$ then, for any r if $r \Longrightarrow_i p$ and $r \in \mathcal{C}_j$ then $r \Longrightarrow_j p$, and for any s if $q \Longrightarrow_i s$ and $s \in \mathcal{C}_j$ then $q \Longrightarrow_j s$.

Proposition 6.3 *The relation $\sqsubseteq^L$ restricted to any well-behaved collection of schemas, $\mathcal{G}_1, \ldots, \mathcal{G}_n$, is a transitive relation.*

Proof: Omitted. ∎

It follows from this that we can find the greatest lower bound of any well-behaved collection of lower weak schemas by "folding" the schemas with the binary greatest lower bound operator, $\sqcap^L$.

Finally we can find the lower merge of such a collection of of schemas by converting their greatest lower bound into a proper schema using an algorithm similar to that described in section 5.2. The only difference between the two algorithms is that, for lower weak schemas, the implicit classes are inserted *above* rather than below the sets of proper classes to which they correspond.

6.4 Other kinds of merges and combined merges

The two alternative kinds of merges, upper and lower, presented in this paper represent two extreme views of what the merge of a collection of schemas should be. It may well be that there are other concepts of merges, perhaps lying between these two, that are equally valid and useful. However it is the author's belief that the methods used here for defining merges are the correct ones, and that any valid concept of merging should have a similar definition in terms of information orderings.

Further it may well be useful to combine different kinds of merges in the process of forming a common view of a number of databases. For example, if we were taking lower merges of a collection of databases we might end up losing a number of specialisation relations which we still believed to be true. We could then re-assert these specialisation relations by taking the upper merge with a new schema containing the relations.

7 Conclusions and Further Work

In this paper we have introduced a simple yet general model for database schemas, and then went on to construct models for the instances of such schemas and for the observations that can be made by querying such databases. There are a number of concepts which exist in the literature of database systems which it might be useful to incorporate in the model. These include such things as cardinality constraints, functional and inclusion dependencies, keys and so on. While we do not believe that it would be difficult to extend the model and the accompanying theory with such concepts, we have opted instead to keep the model relatively simple and easy to formally reason.

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 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.

We went on to describe a second kind of merge which we called *lower merges*. While the *upper merges* defined first represent the sum of the information contained in a collection of schemas, lower merges represent the information common to each schema in a collection. There may well be other, equally valid 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

[Ait84] H. Ait-Kaci. *A Lattice Theoretic Approach to Computation Based on a Calculus of Partially Ordered Type Structures.* Technical Report MS-CIS-84-02, Department of Computer and Information Science, University of Pennsylvania, 1984.

[Ban88] F. Bancilon. Object-oriented database systems. In *Principles of Database Systems*, pages 152–162, 1988.

[BLN86] C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodolgies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, December 1986.

[Che76] 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.

[DH84] 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.

[HK87] R. Hull and R. King. Semantic database modelling: survey, applications and research issues. *ACM Computing Surveys*, 19(3):201–260, September 1987.

[LS83] M. Lenzerini and G. Santucci. Cardinality constraints in the entity relationship model. In C. G. Davis et al., editor, *Entity-Relationship Approach to Software Engineering*, pages 529–549, North-Holland, 1983.

[Mot87] A. Motro. Superviews: virtual integration of multiple databases. *IEEE Transactions on Software Engineering*, Vol. SE-13(7):785–798, July 1987.

[Oho90] A. Ohori. Semantics of Types for Database Objects. *Theoretical Computer Science*, 76:53–91, 1990.

[SBD*81] 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.

[Shi81] D. Shipman. The Functional Data Model and the Data Language DAPLEX. *ACM Transactions on Database Systems*, 6(1):140–173, March 1981.

[SL90] A. Sheth and J. Larson. Federated database systems for managing distributed heterogeneous and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[SLCN88] A. Sheth, J. Larson, J. Cornellio, 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.

[SS77] John Miles Smith and Diane C.P̃. Smith. Database abstractions: aggregation and generalisation. *ACM Transactions on Database Systems*, 2(2):105–133, June 1977.

[WHW90] S. Widjojo, R. Hull, and D. S. Wile. A specificational approach to merging persistent object bases. In Al Dearle, Gail Shaw, and Stanley Zdonik, editors, *Implementing Persistent Object Bases*, Morgan Kaufmann, December 1990.