Using Design Axioms and Topology to Model Database Semantics

Arno Siebes
Martin L. Kersten

Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Abstract

The freedom to combine information stored in a database using the operators provided by its datamodel introduces many caveats, such as with view-updates and integrity preservation, for the database designer. To alleviate these problems we define a formal model that explicates the database semantics through entity definitions and limits their use along well-defined paths. Our approach is based on six design axioms and concepts borrowed topology. This way we achieve an unified description of both the database intension and its extension. In particular, we show that generalisation / specialisation hierarchies are naturally cast into proper subset hierarchies in the entity type topology. Moreover, the limitations posed on the construction of entity types preserve the Armstrong axioms for functional dependencies. This way our model captures much of the real-world semantic constraints and remains sound and complete.

Keywords & phrases: semantic data models, database theory.
1980 Mathematics Subject Classification: 69H21, 22A26, 69K14.
1985 CR Categories: H.2.1, I.2.4.

1. Introduction

Capturing the semantics of a database in a conceptual schema is the prime activity of database design. The focal point of conceptual schema design is how a particular piece of information should be categorised and how it is translated to the concepts provided by the conceptual model. Often it can not be resolved conclusively and uniquely, because the information gathered during the design process is ambiguous and imprecise. One of the most accepted empirical models for database design is the Entity-Relationship model [2] and its variations [5,3]. The important contribution of the EAR model over the relational data model [4] is the distinction between entities (or objects) and relationships among entities (or connections among objects). Relationships in the EAR model deal with semantic properties, such as relationship cardinalities (1:1,1:n,n:m) and existence dependencies, that distinguish them from entities. However, lack of formalisation of the EAR model makes the analysis of a conceptual schema cumbersome.

A more formal approach to database design is pursued in the area of database theory where two main streams can be distinguished: deductive database theory and 'classical' relational database theory. In deductive database theory, logic is used to obtain a proper foundation for modelling database semantics [6,11,7]. The proof-theoretic approach of Reiter [12] shows that indeed many aspects of the relational database model can be formulated as a first-order theory. In particular, it provides a formal treatment of query evaluation in databases with incomplete information (*null* values), the description and enforcement of integrity constraints, and how the relational model can be extended to incorporate more real world knowledge. It's main weaknesses are the reliance on a given conceptual schema and the focus on syntactic aspects. The conceptual schema design process and the semantics being modelled are largely ignored.

The second major stream in database theory is based on the universal relation scheme assumption. One of its main advocates has been Maier [8]. Under the Universal Relationship model the database is defined by a single relation.

Consequently all actions on the database require a projection first. The prime weakness is its lack of rigidity or as Maier puts it: "It all makes sense, if you squint a little and don't think too hard." [Maier83 pg. 371] Furthermore, there is no proper separation between semantics at the intensional level and semantics at the extensional level. This leads to one approach where Maier introduces 'placeholders': members of a set that might not be members of that set after all (sic). In a variation on this approach he uses objects and window functions.

In this paper we propose a new formal model for the description of database semantics †. We start with a set of design axioms that describe the informal concepts attribute, entity, relationship, views and integrity constraints. These axioms are chosen such that the semantic properties recognised as being relevant to the database are named explicitly. Moreover, the axioms disallow the arbitrary manipulation of the attributes to construct user views. Instead, all views should be uniquely decomposable to the underlying semantic primitives. This way view update problems are avoided from the outset. However it this does not rule out that a user sees only part of a view object. It merely ensures that all information to interpret updates are retained by the application program. In addition, the axioms highlight assumptions underlying the older models.

Following we present a formal definition of the database intension, i.e. the allowable entity types in the conceptual schema, using topology. In this approach we show that specialisation and generalisation hierarchies correspond naturally with proper subset hierarchies in the topological space constructed out of the attributes. Since a topological space includes the notion of a (sub) basis, it also provides hints to the database designer as to which entities are really essential and which entities should be considered derivable. Choosing a basis then reflects the bias of the database designer towards the Universe-Of-Discourse.

In the next section we follow the traditional route to define the database extension as a subset of the product space derivable from the attribute domains. The main result, however, is that the relation between database intension and extension can be described within the same formalism. That is, the extension of a database can be seen as a topological space built out of entities rather than entity types. The relationship between database intension and extension then is an injective mapping between two topological spaces. The main benefit is that changes in the database intension can be translated directly into information preserving properties of the database extension. This makes a formal analysis of an evolutionary database schema more tractable.

In the last section we introduce integrity constraints. The focus of our attention is the formal description of functional dependencies. In particular, it is shown how they propagate in the generalisation/specialisation hierarchies, moreover it is shown that functional dependencies behave in a way analogous to extensions. Furthermore, the Armstrong Axioms are captured naturally in our model and we proof that our use of functional dependencies is sound and complete. We conclude with an indication of current and future directions of our research.

† Actually the model is introduced informally; proofs are ommitted.

2. Database modelling axioms

In this section we present the axioms underlying our model and explain how they should be interpreted when modelling part of the real world. The starting point for semantic database modelling is the observation that any model needs a symbolic name space, the non-literals, and value space, the literals [9]. In the database area the symbolic name space is conventionally associated with properties, i.e. perceived distinguishing qualities belonging to an individual or thing. The value space consists of a family of atomic value sets. Moreover, each set of atomic values represents a single semantic concept.

An association of a property name and a value is conventionally called an attribute. It represents a single non-decomposable piece of information extracted from the Universe-Of-Discourse. The property name gives the value in the attribute a specific semantic role. To avoid mis-interpretation one should ensure that an attribute takes an element from a single atomic value set. This leads to the following axiom, present in most database models:

Attribute Axiom:

Each attribute has a single non-decomposable semantic interpretation.

Customary an entity is introduced as a representative for an individual or thing in reality. The properties of the entity are described by attributes while part of the attributes are essential for its identification. We take an opposite position. Namely, we define an entity as nothing more than a name for a set of attributes. Thus the characteristic information of an individual or thing is fully described by its attributes. The entity name itself does not carry additional semantic information.

If we abstract away the value part of attributes, that is we focus on the property set only, then we get an entity type. To simplify identification and manipulation, the designer defines symbolic names for the entity types. Part of the designers' work is to provide all entity types for the database at hand. It is not uncommon that two entity types are defined with an identical property set. Since we take the standpoint that the attributes alone are sufficient to represent an individual or thing, both entity type names should be considered synonyms. Hence one definition can be dropped. If they can not be considered synonyms for the same semantic unit then their attribute sets are underspecified. In that case, the entity type names reveal additional information about the thing being represented. Yet, this information can always be made explicit by an attribute as well. Therefore, to avoid occurrence of semantic information both in the type name and the property list we proclaim that the following axiom should hold for any database design:

Entity Type Axiom:

No two entity types can have the same set of property names.

Since in the Universal Relation approach all its projections are potential entities our entity types form a subset of Maiers objects. More specifically, we ask the database designer to enumerate the semantic meaningful units explicitly to avoid loss of information when a user constructs a view type.

In designing a conceptual model entities are not isolated concepts; rather they participate in relationships. This participation can take many forms. The entities may share a attributes, there may be a functional relationship between attributes, entities may represent components in entity structures, or the system may be informed by the user about a relationship explicitly. In all cases one can consider a relationship as a union of existing entities, augmented with attributes that represent the properties of the relationship. Thus in our view there is no need for a separate relation concept. This reduces the number of primitive concepts to formalise. Moreover, it avoids classification problems encountered during the conceptual schema design. Hence we have the following relationship axiom:

Relationship Axiom:

A relationship is an entity type.

In our model entity types are characterised by their attribute sets, it follows that when two entity types that participate in a relationship have an attribute in common, that attribute occurs only once in the resulting type. Moreover, in that situation a possible instantiation of the entity type is implicitly defined. If this does not comply with the observations from the Universe-of-discourse then it implies that the common attribute has a more complex structure than originally envisioned. For instance, it might be the point at which one discovers that an attribute plays multiple semantic roles or represents an aggregation of smaller entities. But then the attribute axiom us forces to make this information explicit by using a different name for each role.

As mentioned above, we see a relationship as a union of existing entities, augmented with attributes that represent the relationship information. The augmented attributes should play a fairly unimportant role in the relationship. The relationship is determined solely by its contributing entities. In fact, we can generalise this notion to entity types and derive a constraint on the extension of a relationship. Informally, a relationship can not represent information that is not represented by its contributing entities, where the contributing entity types are designated as such by the database designer. This approach will be formalised in the course of this paper. It leads us to the following axiom:

Extension Axiom:

The extension of a compound entity type is fully determined by its contributers.

It is often convenient to combine entity types into clusters and to give them a name for user convenience. Such a construct is called a entity view type. They provide a means to denote semantic units composed of many smaller semantic units. Unlike the older models we restrict view types to sets of entity types. The motivation for this radical step is that now each view is an simple aggregation and all information about its constituents remains available. This limitation ensures that only those views can be constructed for which a unique translation exist for updates. These observations result in the view axiom for database design:

View Axiom:

An entity view type is a set of entity types.

Limitations are often imposed on the actual database states in the form of integrity constraints. These constraints can take many forms, such as limitations on the values in an atomic value set, functional relationship between attributes in a single entity type, as dependencies among entities in the database. In accordance with the relationship axiom it is reasonable to assume that a constraint is defined over existing entity types only. Since they describe part of the real-world semantics, it is mandatory to explicate this information through an entity definition. Therefore, in our opinion dependencies among entities are a generalisation of relationships.

Integrity Axiom:

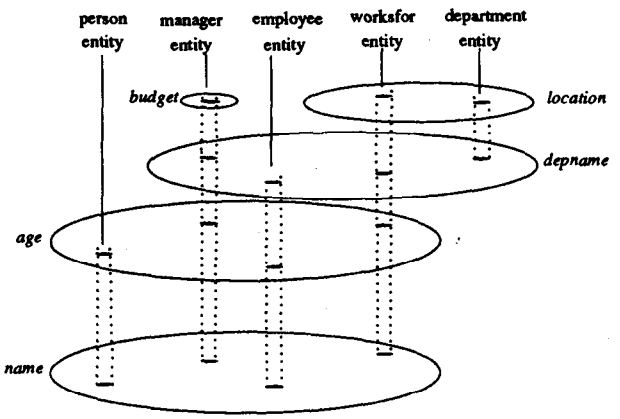
An integrity constraint is a predicate over entity types and implies an entity type.

Our approach to integrity differs from the older models by again shifting the focus to the entities as atoms of information rather than attributes. In this sense, an integrity constraint expresses a desirable property over the (smallest) semantic units, namely entities.

To illustrate the model in the subsequent sections we use the well-known prototype employee database. The semantic distinction between persons' name and departments' name has been made explicit. Integrity constraints such as that "each manager should be an employee", i.e. subset dependencies are represented as subset hierarchies, other constraints are defined later in this paper. The employee database is graphically shown below. This picture visualises the notion that all entities in a database are fully determined by their attributes. In the picture, each attribute corresponds with a disk. Taking a single cut, as shown, results in an instance of an entity type.

A = {name, depname, budget, age, location}
E = {employee, person, department, manager, worksfor}

entity	attribute set
employee	{ name, age, depname}
person	{name, age }
department	{depname, location }
manager	{ name, age, depname, budget }
worksfor	{ name, age, depname, location }



The axioms introduced so far can be used in the database design process to obtain a concise description of the database as follows:

- Derive the property name set, the atomic value sets, and the envisioned attributes from the Universe-Of-Discourse. Use the attribute axiom to ensure that the atomic value set for each attribute is unambiguous.
- Enumerate all entities types, i.e. the entities as found in the Universe-Of-Discourse. When two entity types are indistinguishable from their properties, then they are either underspecified, i.e. additional properties exist, or they play multiple roles. However, the latter can always be resolved through the definition of an additional (role) attribute . The result is a conceptual schema that satisfies the entity axiom.
- If an entity type is an relationship observed in the Universe-Of-Discourse then the common attributes of its contributers should have identical semantic interpretations. Moreover, the relationship axiom requires that relations are defined over entity types only. In particular, the occurrence of common attributes may indicate that the contributing entities are relationships themselves.†
- If the additional attributes in a relationship are needed to identify the relationship occurrences then there should be entity types covering these attributes that have not been made explicit. As, the extension of a relationship is limited by the extension of its contributing entities.
- Remove all entities that are entity views. They can also be constructed from the primitive entities. If, however, this results in loss of information then entity types were missing anyway.
- Dependencies vary over entity types in the context of an entity type (the relation). Thus a dependency might help us in two ways. First we check whether the dependencies varies over entity types. If one of its variables ranges over an attribute only, then, once again, this attribute should be promoted to an entity type. Second we can check whether the implied entity type has been observed as an entity already.

In the next section we will give a more formal description of the database intension, i.e. the database schema, based on the design axioms introduced.

† Or a set of attributes not yet recognised as an entity type.

3. Database intensions

In this section, we impose a topological structure on the entity type space to model the required semantics. In our view the formal description of the database semantics, the conceptual model, starts with the complete list of property names and entity types. This information should come from the database designer; the process by which it is acquired is not of prime interest here. Furthermore, we assume that the above mentioned database design axioms hold. Thus, we start our formalisation process with a finite set $A = \{a_i\}_i$ of property names and a set of entity types $E = \{e_j\}_j$. In particular, each entity type e is a named subset of A: A_e.

In the subsequent sections we will give a formal description of the generalisation/specialisation hierarchy encountered in our conceptual model. Moreover, the role of entities contributing in a relationship is described in more detail. The result of this exercise is that within this framework alternative descriptions of the conceptual model can be formally analysed with respect to preservation of the database semantics.

3.1. The formalisation of specialisation

The database designer may use attributes repeatedly in the description of entities. With each attribute a we can associate the set of entity types V_a in which it is being used, formally

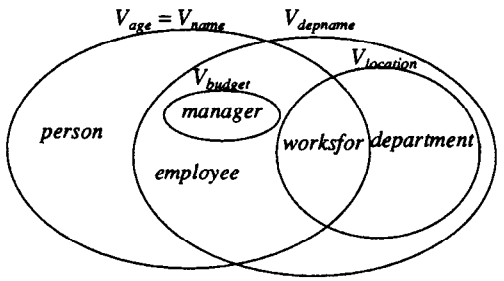
$$V_a = \{e \in E \mid a \in A_e\}.$$

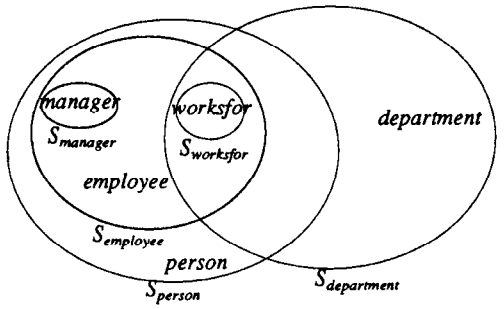
Let V be the family of sets V_a and let L be the set that contains all finite intersections of elements in V. Then for all $e \in E$, L includes a minimal element S_e:

$$S_e = \bigcap_{a \in A_e} V_a = \{f \in E \mid A_e \subseteq A_f\}$$

Alternatively for any W in L, with e as a member, S_e is a subset of W. In the context of a database scheme S_e denotes the set of entity types that are specialisations of e. In fact, e is the root of an ISA-type hierarchy. Conversely, it means that ISA-hierarchies correspond with proper sub-set hierarchies in L, as if $y \in S_x$ and $y \neq x$ then the Entity Type Axiom forces that $x \notin S_y$.

These properties are graphically shown below using a projection of the original disk structure to obtain the more concise ven-diagram.





Since $E = \bigcup_{e \in E} S_e$ it follows that $S = \{S_e \mid e \in E\}$ forms an open cover of E. Obviously, it is the subbase of a topology T and any ISA hierarchy corresponds with a subset hierarchy in this topology. Clearly, S doesn't have to be the smallest subbase. Nor is the subbase per definition unique. It may happen that S contains 'redundant' information. That is, some entity types can be phrased in terms of other entity types using a finite union/intersection expression over elements from the subbase. This gives the freedom to choose a subbase for T which reflects the bias to the Universe of Discourse. Denote by R_T the chosen subbase, the entity types not in the subbase are called constructed types. In our example we have:

$R_T = \{person, department, employee, manager\}$
worksfor is the only constructed element

3.2. The formalisation of generalisation

In the preceding section we have constructed a topology out of attribute sets. It is also possible to define a dual topology based on the attributes omitted in each entity type, and this will lead to a definition of generalisation. Since this will turn out to be an useful topology in its own right, we will actually define it here.

Define $\bar{A}_e = A - A_e$ for each entity type and the family $\bar{V}$ of sets $\bar{V}_a$ as:

$$\bar{V}_a = \{e \in E \mid a \in \bar{A}_e\} = \{e \in E \mid a \notin A_e\}$$

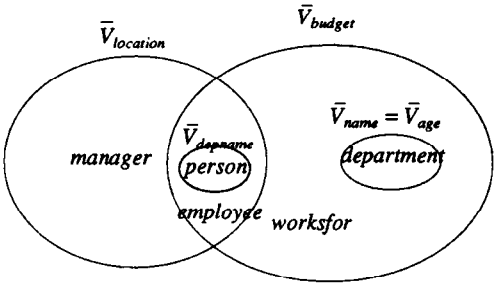
Let $\bar{L}$ be the set that contains all finite intersections of elements of $\bar{V}$. For all $e \in E$, $\bar{L}$ contains the set:

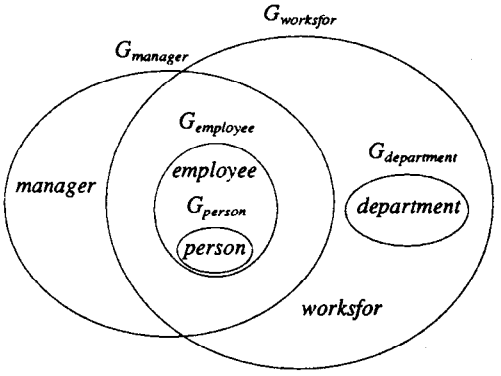
$$G_e = \bigcap_{a \in A_e} \bar{V}_a = \{f \in E \mid \bar{A}_e \subseteq \bar{A}_f\} = \{f \in E \mid Af \subseteq \bar{A}_e\}$$

G_e is the minimal element of $\bar{L}$ that contains e. Interpreted in the context of a database schema G_e denotes the generalisations of e. In particular, let $y \in G_x$ and $y \neq x$ then $G_y \subset G_x$. It is important to remember that S_x and G_x are not each others complements. This would require that $S_x \cup G_x = E$ and $S_x \cap G_x = \emptyset$. A counter example is: $S_{person} \cup G_{person} \neq E$ and $S_{person} \cap G_{person} = person$. However, we do know the following:

Corollary For all $x, y \in E$: $y \in S_x \Leftrightarrow x \in G_y$

Continuing our example, we see:





$E = \bigcup_{e \in E} G_e$ and thus the generalisation sets G_e forms an open cover of E as well, denoted by $G = \{G_e \mid e \in E\}$. Again it generates a topology $\bar{T}$, and once again the subbase used to define it may have redundant entity types and hence we can choose a subbase to reflect our bias.

3.3. Contributers.

Relationships have been recognised before as compound entities, that is, a relationship is represented as union of existing entities and additional descriptive attributes. In fact, every entity that has a generalisation can be seen as a compound entity. This leads to an arbitrary complex for entities and it becomes necessary to explicate the role of the component entities. For this purpose we have introduced the extension axiom, which says that the information in a compound entity is determined by its contributers.
This can be formalised as follows:
Denote by CO_e the set of contributers of e. Then, it is obvious that we want the following property to hold for contributers:
Property If $f \in CO_e$, then $f \in G_e$ and $f \neq x$.

As noted in section two, it is up to the database designer to specify the set of contributers of an entity type. But by choosing the attributes carefully, the designer can achieve that the following definition captures exactly the contributers:

Definition

$$CO_e = \{f \in G_e \mid f \neq e,\ \forall g \in G_e \ s.t.\ e \neq g, f, e \not\subset G_g\}$$

In conclusion we observe that the contributers are the direct generalisations of an entity type.

4. Database extensions

In this section, we formally define the extension of the database. In particular, we show how entities and entity types can be related such that the structure of the entity type space is neatly mapped into the extension space. As a result, we obtain a topological order for the database extension. This provides the means to study alternative physical representations and to analyse the consequences of changes made in the conceptual schema. However, due to space limitation, we describe the intension to extension mapping only.

4.1. Domains

Earlier on we have defined an attribute as an association between a property name (a symbol) and a 'value' (an atomic value). Names are not of prime interest to us in this section. Moreover, we assume that the values are taken from a set of atomic values. In passing we note that when structure is attached to the value sets it becomes possible to introduce *null* values and incomplete information into the model in a natural way, a detailed discussion of this is beyond the scope of this paper. For the time being an attribute value is just a member of a finite set.

Let d_a denote the domain, i.e the set of atomic values, of attribute a. Then the domain of an entity type $e \in E$ is defined as the product of its attribute domains, i.e.

$$D_e = \prod_{a \in A_e} d_a.$$

Furthermore, the set of instances of entity type e, denoted by R_e, is a member of $P(D_e)$. An instance of entity type e, denoted by t_e, is a member of R_e; in the old terminology: R_e is a relation over e and t_e is a tuple in R_e.

The entity type axiom tells us that an entity type is fully determined by its attributes. Thus if we look at a specialisation s of an entity type e and forget about the extra attributes of s, s and e become identical. At the intensional level, this observation is not of much used. But at the extensional level this results in a containment condition on entities. Moreover, it defines an extension mapping as follows:

Definition

Let $e \in E$ and $s \in S_e$, denote by π_e^s the projection $\pi_e^s : R_s \rightarrow P(D_e)$.

The mapping π_e^s projects every $t_s \in R_s$ on D_e. Note that the containment is a direct consequence of the entity axiom; the entities are determined by their properties only. The containment condition on entities is formally defined by:

Containment Condition: $\forall e, s \in E$ such that $s \in S_e : \pi_e^s(R_s) \subseteq R_e$

4.2. Entity type extension

We are now in the position to relate an entity type with the set of allowable instances. Since each extension is a subset of the underlying domain it requires a family of mappings for each entity type. Thus, the extension of an entity type is defined as follows:

Definition

The mapping $E_e : S_e \rightarrow P(D_e)$, maps $s \in S_e$ to $\pi_e^s(R_s)$.

Observe that with this definition we take care of the situation that information about entity type instances might be 'stored' within its specialisations only. Moreover, the mapping from database intension to extension functions as an integrity constraint on the allowable database states, i.e. the mappings only allow extensions within the appropriate domains. Furthermore, they allow us to define the extension as a topological space, but, once again, this is beyond the scope of this paper.

The definition of the E_e allows us to give a formal description of the extension axiom. The axiom requires that the information contained in a relationship does not exceed the information obtainable from its contributers. Thus we need an operation with wich we can combine the information in the various contributers, this operation is of cause the well known join, which we denote by * or Π if we take the join of more than two sets. Now the extension axiom is rephrased as follows:

Extension Axiom :

$i : E_e(e) \rightarrow \prod_{c \in CO_e} E_c(e)$ if CO_e is nonempty, where i is an injective function.

We defined their to be an injective mapping instead of requiring $E_e(e)$ to be a subset of the above join because e might have extra attributes. The injectivity means that when we choose an entity e_i for every entity type in CO_e, this combination of entities $(\{e_i\})$ can form at most one entity of type e. For example, an employee can be a manager in at most one way.

We'll end this discussion with a definition and a useful corollary:

Definition

Denote by $\rho(h,f,e)$ the mapping $E_e(h) \rightarrow E_e(f)$, for $S_h \subseteq S_f \subseteq S_e$.

The definitions and the containment condition immediately imply:

Corollary

If $S_h \subseteq S_f \subseteq S_e$, then

a $\pi_e^f \pi_f^h = \pi_e^h$

b $\rho(f,e,e)\rho(h,f,e) = \rho(h,e,e)$

c $\pi_e^f \rho(h,f,f) = \rho(h,f,e)\pi_e^f$

It should be noted that the containment condition translates the ordering of entity types reached at the intensional level to the extensional level. Now that we have formally defined extensions and their relation to the intensional level, we can continue with dependencies.

5. Integrity constraints

An essential part of a conceptual schema is the description of the relevant integrity constraints. Often, integrity constraints are the only means to model real-world semantics in the database in a concise and formal way. The unattractive alternative being dispersion of these checks over the application programs. Therefore, a vast amount of dependencies have been defined in database theory. In this section we study the role of functional dependencies in the context of our model. Studying functional dependencies suffices to capture the essence of dependencies in our model, moreover a treatment of other dependencies is far beyond the scope of this paper.

Recall the integrity axiom, which states that integrity constraints vary over existing entity types in the context of another entity type. This means that dependencies are not formulas over attributes but over entity types. Moreover, they are only meaningful if there exists a context, i.e. there exists an entity type which is a specialisation of all the entity types involved. Note that the context is necessary to disambiguate dependencies as well. Since entity types may be related in several ways.

5.1. Functional Dependencies

Functional dependencies are the most thoroughly studied dependencies in database theory. An entity B is functional dependent on A in a relation if in every tuple of the relation $R(A,B,.....)$ in which we encounter a specific value a_1 for A, we will always find the same value, say b_1 for B; thus an A can be associated with at most one B. The translation to our model is straight-forward:

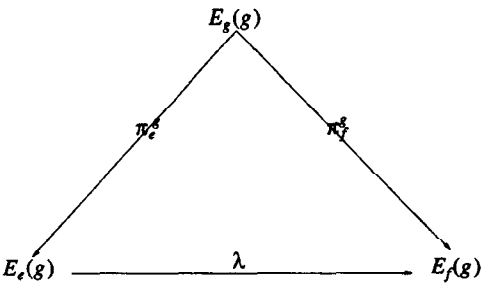
Definition

Let $e,f,h \in E$ such that $e,f \in G_h$ e functional defines f in the context of h, denoted $fd(e,f,h)$ if: $\forall R_h, \forall t_h^1, t_h^2 \in R_h$: $\pi_e^h(t_h^1) = \pi_e^h(t_h^2) \rightarrow \pi_f^h(t_h^1) = \pi_f^h(t_h^2)$

This definition can be visualised as follows:

Theorem

Let $e,f,g \in E$ such that $e,f \in G_g$, then $fd(e,f,g)$ iff $\forall R_g \exists \lambda: E_e(g) \rightarrow E_f(g)$ such that the following triangle commutes:



5.2. Armstrong Axioms

The basis for most results obtained in the theory of functional dependencies is of course the Armstrong Axioms [1] One way to phrase them is:

1 $\forall i \in \{1..m\}\ A_1A_2.....A_m \rightarrow A_i$.

2 $A_1A_2.....A_m \rightarrow B_1B_2.....B_r$ **iff** $\forall i \in \{1..r\}\ A_1A_2.....A_m \rightarrow B_i$.

3 If $A_1A_2.....A_m \rightarrow B_1B_2.....B_r$ and $B_1B_2.....B_r \rightarrow C_1C_2.....C_p$ then $A_1A_2.....A_m \rightarrow C_1C_2.....C_p$

We can rephrase these axioms in our model as follows:

Armstrong Axioms

1 $\forall g \in G_e : fd(e,g,e)$.

2 $fd(f,g,e)$ iff $\forall h \in G_g\ fd(f,h,e)$.

3 If $fd(f,g,e)$ and $fd(g,h,e)$ then $fd(f,h,e)$

Note that 2 is sound because of the Extension Axiom.

The Armstrong axioms give a locally sound and complete system, locally because dependencies extend via the ISA hierarchies in a way that is not captured by the axioms:

Theorem

Let $e,f,g \in E$ such that $e,f \in G_g$ and $fd(e,f,g)$, furthermore let $h \in S_g$ then $fd(e,f,h)$ also holds.

And now we have a global sound and complete system:

Theorem

The Armstrong Axioms, together with the propagation theorem are a sound and complete system.

5.3. Dependency Mappings

Above we have seen that functional dependencies propagate just as extensions. This similarity can be used to define a mapping connecting entity types to functional dependencies. Before doing so we should define an appropriate domain for the resolving entity type. This domain should satisfy the Armstrong Axioms. Moreover, if the context e is known, $fd(x,y,e)$ can be denoted by (x,y), i.e. the fd's in the context of e are a subset of $G_e * G_e$. These requirements lead us to the following approach.

Denote by N_e, the nucleus of e, those fd's that should always hold in G_e, i.e. N_e is the smallest set such that:

$$\forall x \in G_e, \forall y \in G_x, (x,y) \in N_e.$$

Denote by F_e the following set:

$$F_e = \{y \in P(G_e * G_e) \mid N_e \subseteq y\}.$$

And finally denote by F_e^*, the transitive closure of the elements of F_e under the third Armstrong Axiom; i.e. let $y \in F_e$ and $(a,b),(b,c) \in y$ then y^* contains also (a,c).

Definition

The domain for functional dependencies over e, DF_e is: $DF_e = F_e^*$

Denote by fd_e that element of DF_e which we want to hold. Then the propagation theorem tells us that $fd_e \subset fd_f$ for $f \in S_e$. But $fd_f \cap DF_e$ might be a superset of fd_e as their may be functional dependencies between elements of G_e in the context of f that are not valid in the context of e.

This leads to the following definition:

Definition

The mapping $F_e : S_e \rightarrow DF_e$, is defined by: $F_e(f) = fd_f \cap DF_e$

Note that in general $F_e(f)$ is not closed under the Armstrong Axioms because (f,e) is not an element. We can mimic the extensions even more, by defining:

Definition

Let $S_g \subseteq S_f \subseteq S_e$, then

1 $\rho F(f,g,e)$ denotes the mapping: $F_e(f) \rightarrow F_e(g)$.

2 πF_f^e denotes the mapping: $F_e(g) \rightarrow F_f(g)$.

And this gives us the corollary:

Corollary

If $S_g \subseteq S_f \subseteq S_e$, then

a $\pi F_g^f \pi F_f^e = \pi F_g^e$

b $\rho F(f,g,e) \rho F(e,f,e) = \rho F(e,g,e)$.

c $\pi F_f^e \rho F(f,g,e) = \rho F(f,g,f) \pi F_f^e$.

So again we translated the ordering reached at the intensional level to an ordering at a different level, the database extension.

6. Summary and future research

In this paper we have introduced a new formal model for the description and analysis of database semantics. Our approach differs from earlier attempts by presenting a concise set of design axioms and using mathematical well-established concepts. The main results are summarised as follows. It is shown that the database intension can be cast in a topological space constructed out of attributes. From this we can derive the extension, the possible database states, through well-defined mappings.

Entities in this topological space are names for attribute sets. They do not bear any additional semantic information from the real-world being modelled. This approach is reminiscent of the approach taken by Maier, but, in contrast, semantics play a more fundamental role in our approach. The user is limited in the way entities can be composed to for views. We only allow a user to combine entities such that their is always a proper translation back to its constituents. This way it avoids the view-update problems encountered in other approaches where the projection operator can easily destroy the semantic bonds between attributes composing an entity.

Currently we investigate more complex constraints, such as multi-valued dependencies, join-dependencies and domain constraints. It can be shown that multi-valued dependencies are a special case of domain constraints. Imposing a structure on the domain, a boolean algebra structure [10], results in a formal definition of *null* values and incomplete information. It differs from the method proposed by Reiter where the interpretation of the *null* is context dependent and affects the definition of functional dependencies. In our approach, the *null* interpretation can be defined independent of the entity type structure and it semantics carry over to functional dependencies.

Since both extension and intension are cast into a single formalism and their relationship can be formally described by functions. In particular, we use sheaf theory [13] to study the continuity problems in databases, i.e. updates of both intension and extension. Results on these issues will be published in forthcoming papers.

References

[1] Armstrong, W. W., "Dependency Structures of Data Base Relationships," *Proc. IFIP Congress 1974.*

[2] Chen, P.P., "The entity-relationship model: towards a unified view of data," *ACM Transactions on Database Systems*, vol. 1, no. 1, pp.9-36, 1976.

[3] Chen, P.P., *Entity-Relationship Approach to Systems Analysis and Design.* Amsterdam:North-Holland, 1983.

[4] Codd, E.F., "A Relational Model for Large Shared Data Banks," *Comm. ACM*, vol. 13, no. 6, pp.377-387, 1970.

[5] Elmasri, R., Weeldreyer, J., and Hevner, A., "The category concept: An extension to the entity-relationship model," *Data & Knowledge Engineering*, vol. 1, no. 1, pp.75-116, 1985.

[6] Gallaire, H. and Minkers), J., in Logic and Databases, Plenum Press, New York (1978).

[7] Gallaire, H., Minker, J., and Nicolas, J-M., "Logic and Database: A Deductive Approach," *ACM Computing Surveys*, vol. 16, no. 2, pp.153-185, June 1984.

[8] Maier, David, "Null Values Partial Information and Database Semantics," pp. 371-438 in The Theory of Relational Databases (1983).

[9] Nijssen, G.M., "The binary relationship approach," in Concepts and Terminology for the Conceptual Schema and the Information Base, ed. J.J. van Griethuysen (1982).

[10] Raisiowa, H. and Sikorski, R., *Mathematics of Metamathematics.* Polish University Press.

[11] Reiter, R., "Databases: a Logical Perspective," *Proc. Workshop on Data Abstraction, Databases, and Conceptual Modelling. SIGPLAN Notices*, vol. 16, no. 1,

pp.174-176, Jan 1981.

[12] Reiter, R., "Towards a Logical Reconstruction of Relational Database Theory," pp. 191-233 in On Conceptual Modelling, ed. J.W. Schmidt (1984).

[13] Tennison, B. R., *Sheaf Theory*. Cambridge University Press.