1985

Generalization Per Category: Theory And Application

Mohammad A. Ketabchi
University of Minnesota

Valdis Berzins
University of Minnesota

Kurt Maly
University of Minnesota

Generalization Per Category: Theory And Application

Mohammad A. Ketabchi
Valdis Berzins
Kurt Maly
Computer Science Department
University of Minnesota
Minneapolis, MN 55455

ABSTRACT

The concept of Generalization Per Category (GPC) is formalized. It is shown that GPC imposes lattice structures on entity types and their subtypes. A high level application oriented data definition language based on the GPC is outlined which allows the system to derive general entity types and organize their instances. Users are freed from undue efforts in the design of databases which are about entity types with rich varieties and high populations. Effective browsing of these databases and efficient execution of frequent queries against them are achieved by using the lattice structures among the entity types and their subtypes.

Introduction

Because of the limitations of human intelligence in dealing with the many entities that exist in large databases, most data models impose structure on the database by grouping these entities into entity types. The relational model, hierarchical model, and network model are examples of such data models called *Strictly Typed Data Models* (STDM) (Tsichritzis, 1982) or *formated data models* (Codd, 1979). STDM requires each datum to belong to some type. Data that do not naturally fall into a type must be subverted to fall into one. In the relational data model for example, a relation represents an entity type which is a group of similar entities. Similarities between entities result in common properties which appear as the attributes of the relation.

Similarity is relative to the level of abstraction. Two things which are similar in a higher level of abstraction might be dissimilar in a lower level of the abstraction. If the modeler insists on keeping two entities in the same group at a level of abstraction in which the differences are exposed, *property inapplicable null values* (Vassiliou, 1979) must be allowed. Null values are source of many difficult problems which are generally not well understood (Date, 1983).

When the dissimilarities among entities in a group become significant it is divided into smaller groups. In the relational model this results in the loss of the semantics of the data because logically similar entity types are now represented by different relations. The model does not capture the fact that in a higher level of abstraction they are the same type.

Many researchers have investigated this problem. Consensus are that *generalization abstraction* is the proper solution (Kent, 1979 and Tzichritzis, 1982). A model which supports generalization will allow the modeler to view the similar entity types as one type in the higher level and as different types in the lower level of abstraction. The inverse of generalization is *specialization.* If entity type G is a generalization of entity type S, then S is a specialization of G. We will refer to G as a *generalized type* and to S as a *specialized type.*.

Entity types are Cartesian aggregations of different property types. Each property type can be generalized to a supertype. Therefore, each entity type can be generalized according to different properties. When the generalization of entities per different proerties are allowed, the number of entity types increases rapidly. Codd (Codd, 1979) and Smith and Smith (Smith, J., and Smith, D. 1977-2) distinguish the properties which are used for generalizations and call them categories. We call them *categorical properties* to distinguish them from categories of entities. Smith and Smith do not allow the same entity type to belong to generalizations per more than one categorical property at a time. In their model the generalization hierarchy is a tree. Codd allows generalization per category in RM/T and represents the structure among subtypes of a general entity type by a directed graph. The next Subsection provides more information about the concept of generalization and its evolution.

The complicated structure among the generalizations of entities per different categorical properties and in different levels tends to offset the intellectual manageability that can be gained by using the concept of generalization.

If the concept is to be a useful organizational tool in real world situations the system must accept more responsibility. Users should not be required to specify all the different types, supertypes, and their properties because there will be too many of them. The amount of information provided by the users should be reduced and the role of the system in the generalization process must be enhanced. This paper reports an attempt to achieve this objective.

In the next section we study the categorical properties of entity types independently and show that the generalizations of each categorical property form a lattice structure. We then show that possible generalizations of Cartesian aggregation of different categorical properties form a lattice structure which is the direct product of the lattices of relevant properties. In the following section we show that generalizations of entity types per different categories form a structure which is isomorphic to the lattice of the categorical properties. We then outline a user interface for a DBMS which supports GPC as developed in the previous sections. We describe how the lattice of GPC can be used for effective browsing of databases and efficient retrieval of data. The application of GPC in the development of management information systems (MIS) is discussed in this section also. Finally, we suggest directions for future work and offer concluding remarks.

BACKGROUND AND PREVIOUS RELATED WORK

The intellectual manageability of large databases can be enhanced by imposing structure on their object population. Classification and generalization have been extensively used for this purpose. Classification is grouping the similar entity instances into entity types. Generalization is grouping similar entity types into higher level types. The similarities are emphasized by considering them as attributes of the higher level types. Structure among the entity types and the higher level types are usually restricted to a hierarchy of predefined nodes and is called a *generalization hierarchy* (Tsichritzis, 1982 and Smith J., Smith D., 1977-2).

The extensional aspect of generalization is inclusion (subset). The set of the entities of a specialized entity type is a subset of the set of entities of the generalized type. The intentional aspect of generalization is that the properties of generalized types are inherited by their specializations. It is this aspect of the generalization which allows property inapplicable null values to be avoided. Note that entity types might have attributes which apply to the type as a whole rather than instances of the type. These properties are not inherited.

The notion of generalization has received considerable attention in the context of semantic nets in artificial intelligence, and semantic data models in databases. Smith and Smith have applied the concept to formated databases as a database design tool. They restrict the relationships among generalized and specialized entity types to be trees. This implies no entity type can be a specialization of more than one entity type. As an example entity type MALE-ENGINEER can not be generalized to MALE-EMPLOYEE and ENGINEERS at the same time. In the real world, objects are usually generalized according to different categories and the requirement of Smith and Smith is not acceptable. Codd in RM/T (Codd, 1979), Mylopoulos, Bernstein, and Wong in TAXIS (Mylopoulos, 1980), and Hammer and McLeod in SDM (Hammer, 1981) lift the restriction of strict hierarchy and allow *generalization per categories*. Mylopoulos calls the structure among entity types (called classes in TAXIS) IS-A hierarchy. Codd calls it generalization graph.

In TAXIS and RM/T users are responsible for declaring the entity types and managing the existing relationships among subtypes and their generalizations. If users have not declared the entity type ENGINEER as the generalization of MALE-ENGINEER and FEMALE-ENGINEER per category SEX the entity type ENGINEER does not exist in the database. This is acceptable if there are very few categories, very few levels of abstractions, and very few meaningful combinations of subcategories. There are applications which do not satisfy these restrictions. One example is parts databases for CAD applications. Another example is the information architecture of management information systems of large organizations.

Consider a CAD parts database where PARTs have POWER-CONSUMPTION, SPEED, and OPERATION categories. POWER-CONSUMPTION is a generalization of LOW-POWER and HIGH-POWER, SPEED is a generalization of FAST and SLOW, OPERATION is a generalization of LOGICAL-OP and ARITHMETIC-OP, LOGICAL-OP is a generalization of AND, OR, and NOT operations, and ARITHMETIC-OP is a generalization of ADD and SUB operations. The following is a partial list of subtypes of entity type PART:

LOW-POWER PART
HIGH-POWER PART
FAST PART
SLOW PART
FAST-LOW-POWER PART
SLOW-LOW-POWER PART
FAST-HIGH-POWER PART
SLOW-HIGH-POWER PART
FAST-AND PART
LOW-POWER-AND PART
SLOW-LOW-POWER-OR PART
FAST-LOGICAL-OP PART

Although this example has been simplified significantly it shows the complexity of the modelers' task if they want

to take full advantage of the concept of generalization. Unless the system provides proper help and facilities, users will prefer not to use GPC, or they will impose artificial restrictions on their data. We address this problem and develop a solution which allows the use of generalization without undue design effort and artificial restrictions.

We extend previous work in two ways: (i) We discover and formalize the precise structure among generalized entity types and their specializations. This structure is a complete lattice, isomorphic to the direct product of the lattices of categorical properties. This is the major theoretical contribution of this paper. (ii) Once the structure among generalized types is known, database management systems that can take more responsibilities in the generalization process can be designed. We demonstrate this by outlining a user interface which requires users to provide a reasonable amount of information only. The system can then create the generalized types, partition attributes among them, and supervise the inheritance rules. This is the major practical contribution of this paper.

THE PROBLEM

Consider entity type E. Let A_i, for all $i \in \{l \mid l = 1..n\}$ be the attributes of E. Let C_j, for all $j \in \{l = 1..m\}$ be the categorical properties of E. Each categorical property C_j may have elements. If we view a categorical property as a data type then its elements correspond to subtypes of that data type. Instances of entity type E can be grouped according to combinations of the elements of its categorical properties. These groups can be defined as entity types themselves. Let T be the set of these types and the entity type E. Let P and $Q \in$ T. Define the binary relation, generalization-of ($\leq_{gen}$), as follows:

$P \leq_{gen} Q$ *iff*
(i) any instance of Q is an instance of P; and
(ii) Q inherits all the attributes of P;

Define specializtion-of ($\geq_{gen}$) as $Q \geq_{gen} P$ iff $P \leq_{gen} Q$.

We are interested in finding acceptable, that is useful, feasible, and correct answers for the following questions:

(1) What are the precise structure and relationships among the specializations and generalizations of entity type E?
(2) How are the attributes A_i assigned to the specializations of E in order to suppress irrelevant details and avoid inapplicable null values?
(3) Generalization is an organizational principal in users' level. How can users define and manage all the meaningful generalizations and specializations of entity type E per different categories?

LATTICES OF CATEGORICAL PROPERTIES

Let C_j be a categorical property of an entity type E. C_j can be viewed as an enumeration data type. We call the elements of the enumeration, *categorical elements of* C_j. Categorical elements can be enumeration types themselves and in this respect they are categorical properties. No categorical element may be an element of more than one categorical property. This restriction guarantees that the hierarchies of categoricial properties are trees. If we allow a categorical element ot have two parents there will be no way to recognize different generalizations of the same entity type per the same category. We believe that whenever an element of a categorical property tends to have more than one generalization there is a hidden categorical property. By hidden categorical property we mean an undeclared property that exists in users' logical view. To define the hierarchies of categorical properties we will use a Pascal like syntax for enumeration data types. Two examples follow:

Example-1:
categorical properties:
JOB = (ENG, SEC)

Example-2:
categorical properties:
OP = (ARITH, LOG)
ARITH = (ADD, SUB)
LOG = (OR, AND, NOT)

We add a *bottom element*, denoted by !, to the hierarchies of categorical properties. This element is the categorical element of all the leaves of the tree. We will refer to the leaves of the tree as *atoms* of the categorical property designated by the root. Figure 1 shows the hierarchies of categorical properties JOB and OP with the bottom element added.

In the next subsection we show that the set of the nodes in the hierarchy of a categorical property and the bottom element have the structure of a complete lattice. We will then extend this structure to Cartesian aggregation of multiple categories and will use it to organize generalizations of entity types per different categorical properties.

THE LATTICE OF A SINGLE CATEGORICAL PROPERTY

Let S_j be the set of the bottom element and all the nodes in the hierarchy of category C_j. Let X, Y, and $Z \in S_j$.

Let PARENT-OF be a binary relation on S_j such that X PARENT-OF Y iff Y is an element of the enumeration of X. Then the root of the hierarchy has no parent and the bottom element is the PARENT-OF no other element.

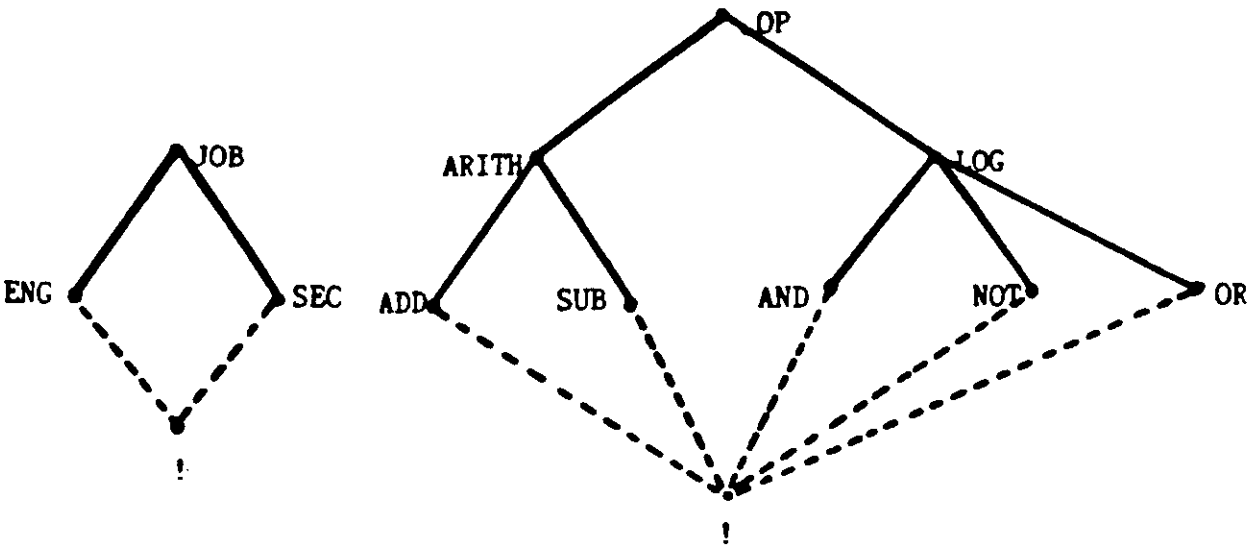

Figure 1

Let $\leq_j$ be the reflexive transitive closure of the PARENT-OF relation. Then $\leq_j$ is reflexive, transitive, and antisymmetric, and hence a partial order. Define $\geq_j$ by $X \geq_j Y$ iff $Y \leq_j X$. The relationship $X \leq_j Y$ means that either X and Y are the same element, or Y is descendant of X at some level of the hierarchy of C_j.

Referring to category OP as an example (see Figure 1), the following relationships are true:

ARITH PARENT-OF ADD
OP $\leq_{op}$ ARITH
OP $\leq_{op}$ ADD
NOT $\geq_{op}$ LOG
LOG PARENT-OF NOT
LOG $\leq_{op}$!

Let ANCESTORS: $Y \rightarrow \mathbf{R} \subseteq \mathbf{S}_j$ be a mapping on $\mathbf{S}_j$ such that $X \in$ ANCESTORS (Y) iff $X \leq_j Y$. Then the only ANCESTORS of the root is the root itself and every element is ANCESTORS of the bottom element.

Let the CLOSEST-ANCESTOR: $\mathbf{R} \subseteq \mathbf{S}_j \rightarrow Y$ be a mapping on $\mathbf{S}_j$ such that Y = CLOSEST-ANCESTOR ($\mathbf{R}$) iff (i) $Y \in$ ANCESTORS (Z) for all $Z \in \mathbf{R}$, and (ii) there does not exist a $U \in \mathbf{S}_j$ such that $Y \in$ ANCESTORS (U) and $U \in$ ANCESTORS (Z) for all $Z \in \mathbf{R}$. Note that the CLOSEST-ANCESTOR of a set with one element is the element itself. Referring to example-2:

ANCESTORS (OR) = {LOG, OP, OR}
ANCESTORS (!) = {!, ADD, SUB, OR, AND, NOT, ARITH, LOG, OP}
CLOSEST-ANCESTOR ({ADD, OR}) = OP
CLOSEST-ANCESTOR ({AND, OR}) = LOG
CLOSEST-ANCESTOR ({ARITH, ADD}) = ARITH
CLOSEST-ANCESTOR ({!}) = !
CLOSEST-ANCESTOR({!, NOT}) = NOT
CLOSEST-ANCESTOR ({ADD, LOG}) = OP

A lattice is a partially ordered set with two operations of join and meet. We define the operations join and meet on the set $\mathbf{S}_j$, denoted by $+_j$ and $*_j$ respectively, as follows:

$X +_j Y$ = CLOSEST-ANCESTOR ({X, Y})
$X *_j Y = X$ if $X \geq_j Y$, else
Y if $Y \geq_j X$, else
!

These operations will be extended and used in the next section to find generalizations and specializations of entity types.

Theorem:: The partially ordered set $< \mathbf{S}_j, \leq_j >$ is a complete lattice whose join and meet operations are $+_j$ and $*_j$.

The proof follows directly from the definitions of $\leq_j$, $+_j$, and $*_j$. We will refer to this lattice as *single category lattice* of categorical property C_j and will denote it by L_j. The single category lattices JOB and OP of examples 1 and 2 are shown in Figure 1.

THE LATTICE OF MULTIPLE CATEGORICAL PROPERTIES

In general, entity types have more than one categorical property and the instances can be grouped according to combination of the elements of these properties. Therefore, we study the relationships among Cartesian aggregations of categorical properties.

Let C_j, for all $_j \in \{l \mid l = 1..m\}$ be the categorical properties of entity type E. Let $L_j = < \mathbf{S}_j, \leq_j, +_j, *_j >$, be the lattice of categorical property C_j. Define $\mathbf{S} = \prod_{j=1}^{m} \mathbf{S}_j$ ($\prod$ stands for repeated Cartesian product). The members of $\mathbf{S}$ are m-tuples of X_j, for all $_j \in \{l \mid l = 1..m\}$, where $X_j \in \mathbf{S}_j$. Let X and $Y \in \mathbf{S}$.

Define $\leq$ where $X \leq Y$ iff $X_j \leq_j Y_j$ for all $j \in \{l \mid l = 1..m\}$. $\leq$ is a partial order relation in S. The partially ordered set $< \mathbf{S}, \leq >$ is a complete lattice in which the join and meet operations are defined as follows (proof is trivial):

$$X + Y = < X_1 +_1 Y_1, X_2 +_2 Y_2, \ldots X_m +_m Y_m >$$
$$X * Y = < X_1 *_1 Y_1, X_2 *_2 Y_2, \ldots X_m *_m Y_m >$$

The bottom of the lattice $< \mathbf{S}, \leq, +, * >$ is: $_! = < !_1, !_2, \ldots !_m >$. The top of the lattice is the m-tuple of tops of the lattices L_j. We call this lattice *multiple category lattice* of the entity types with categorical properties C_j and denote to it by L.

Since $!_j$ is an element of $\mathbf{S}_j$, then their Cartesian product $\mathbf{S}$ will contain nodes that have $!_j$, for some $j \in \{l \mid l = 1..m\}$ as their components. We call these elements of $\mathbf{S}$ *incomplete elements*. Intuitively, these elements represent the objects which belong to more than one categorical elements within the same categorical property. Since we have disallowed these elements, the existence of objects corresponding to incomplete elements indicate error conditions.

The Lattice of Generalization Per Category

In this section we show that the structure of generalizations of entity types per categorical properties C_j, for all $j \in \{l \mid l = 1..m\}$ and in different levels is isomorphic to the lattice L defined in previous section. We call this lattice GPC lattice. In the next section we show how this isomorphism can be used in database definition and manipulation, and development of information architecture.

We have made the following assumptions:

(i) Entities have unique identifiers. We suggest surrogate keys (Kent, 1979) that are defined and controlled by the system and are transparent to the users.

(ii) The categorical properties of all the instances are known.

(iii) An attribute which is inapplicable to an instance e is inapplicable to all other instances with the same elements of categorical properties as e. As an example, if attribute TYPING__SPEED is inapplicable to a MALE__ENGINEER then it is inapplicable to all MALE__ENGINEERs.

THE CONSTRUCTION OF GPC LATTICE

In the following construction we form entity types corresponding to the nodes of the multiple category lattice L. We determine the attributes and instances of these entity types. We then prove that the set of these entity types is isomorphic to the lattice L.

(1) Let $\mathbf{I}$ be the set of all entity instances of type E: Divide the set $\mathbf{I}$ into groups of instances, such that there is one group corresponding to each combination of the atoms of categorical properties C_j. If there are p atoms per categorical property, then there will be p^m groups formed in this step. Assumption (ii) implies each entity instance will belong just to one group. Consider each group as a distinct entity type. These entity types are the most specialized entity types. We will refer to these entity types as *min-entity types*.

(2) Step 1 populates the min-entity types by their instances. In this step attributes of these entity types are determined. Let $\mathbf{T}$ be the set of all min-entity types. Assign attributes A_i, for all $i \in \{l \mid l = 1..n\}$, to entity types in $\mathbf{T}$ such that each entity type is assigned applicable attributes only. That is, if A_k is not applicable to entity type $U \in \mathbf{T}$ then U does not have attribute A_k. Assumption (iii) implies that it will never be the case that A_k is applicable to some instances of U but not to all of them.

(3) In this and the next step we form new entity types correspnding to those elements of S which are not Cartesian product of atoms. To do this we start with the elements which correspond to min-entity types and repeatedly use + operation until there is an entity type for each element in S (except incomplete elements). Let Z and U_r for all $r \in \{l \mid l = 1..q\}$ be elements of S, such that

$$Z = U_1 + U_2 + .. + U_4 = \sum_{r=1}^{q} U_r.$$

If no $Z_t \in$ T corresponding to $Z \in$ S has been formed yet, but an entity type corresponding to each U_r has been defined in T, move all the surrogates in U_r, for all $r \in \{l \mid l = 1..q\}$ into a new group. Consider this group a new entity type Z_t and assign $\mathbf{T} \cup Z_t$ to T.

(4) Repeat step 3 until no new entity type can be added to T. Note that there will be an entity type which contains all the instances in I and is the highest level entity. This entity type corresponds to top of the lattice L and is entity type E. We will refer to this entity type as *max-entity type*.

(5) In this step we determine the attributes of the entity types formed in steps 3 and 4 by using the attributes of min-entity types. Start with the min-entity types defined in steps 1 and 2. Let $Z_t \in$ T correspond to $Z \in$ S. Find the common attributes among all entity types corresponding to U_r, for all $r \in \{l \mid l = 1.. q\}$,

where $Z = \sum_{r=1}^{q} U_r$, and designate them as attributes of Z_t. Repeat this step until attributes of all entity types in T are determined.

We now have a set of entity types T whose elements are groups of entities described by the same entity types. If we proof this set is isomorphic to the lattice L we can use the previous construction to form the GPC lattice and we can use + and * operations to calculate the generalizations and specializations of entity types.

Theorem: The set T is a complete lattice isomorphic to lattice L.

proof: we need to show that (a) there exist a bijective mapping from the set T to the set S, and (b) the join and meet operations, homomorphic to + and * in L, can be defined in T.

(a) Steps 1 and 3 guarantee that there exists an element in T corresponding to each element of S and vice versa. In order to make the mapping of T to S bijective we need to define an entity type in T corresponding to each incomplete element of S. These entity types will have no instance because we have excluded the entities which belong to more than one categorical element within the same categorical property. Mathematically speaking, the attributes of these entity types are the union of the attributes of their generalizations. In implementation these nodes can be handled easily because they do not enter the normal operation of the database.

(b) Recall the definition of $\leq_{gen}$. Let U_t, V_t, $Z_t \in$ T. We use $\leq_{gen}$ (see subsection "The Problem") to define the following operations on set T:

Define $+_{gen}$ as $Z_t = U_t +_{gen} V_t$ iff: (i) $Z_t \leq_{gen} U_t$, and (ii) $Z_t \leq_{gen} V_t$, and (iii) there exist no $W_t \in$ T with properties (i) and (ii), where $Z_t \leq_{gen} W_t$.

Define $*_{gen}$ as $Z_t = U_t *_{gen} V_t$ iff: (i) $Z_t \geq_{gen} U_t$, and (ii) $Z_t \geq_{gen} V_t$, and (iii) there exists no $W_t \in$ T with properties (i) and (ii) where $Z_t \geq_{gen} W_t$.

We show that $+_{gen}$ and $*_{gen}$ in T are homomorphic to + and * operations in S.

Let Z, U, and $V \in$ S. Let $Z = U + V$. Step 3 guarantees that there exists a Z_t such that condition (i) of $\leq_{gen}$ is satisfied for $Z_t \leq_{gen} U_t$ and $Z_t \leq_{gen} V_t$. Step 5 guarantees the condition (ii) of $\leq_{gen}$. The uniqueness of $Z \in$ S implies that $Z_t \in$ T is unique, therefore, given U and $V \in$ S and their corresponding elements U_t and $V_t \in$ T, $U + V \in$ S corresponds to unique $U_t +_{gen} V_t \in$ T. Therefore, correspondence of elements in S and entity types in T, as established in steps 1 and 3 preserves the join operation. A similar argument can be made for meet operation.

The GPC lattice can be exploited for several practical purposes as described in the next section.

Using the Lattice of Generalization Per Category

We perceived the usefulness, and the difficulties of generalization per category in the design and use of databases which have the following structural and operational characteristics:

(i) The database is about a general entity type which can be categorized in several ways and can be divided into different subtypes. The general entity type represented by the database is the max-entity type. Subtypes are the spe-

cializations of this entity type per different categories.

(ii) The main operation is browsing the database to find and retrieve entities which meet certain desired properties as closely as possible. Browsing is exploratory search of the database where the object of the search is specified by the elements of categorical properties. The goal is to approximate the desired object if it does not exist in the database, by another object in the database. The set of elements of categorical properties might be a complete set (that is, there is an element for each categorical property), or a partial one (that is, no element has been specified for some categorical properties). If the search starts with complete set and fails it should be *broaden*. This means the generalizations of the entity type which was checked should be searched. If the search starts with a partial set and several entities are found then it should be *restricted*. This means that the specializations of the entity type which was checked must be searched.

(iii) The database is very large and contains a variety of information used by diverse groups of users. An example is information systems of large organizations where there are several categories of information with potentially many subcategories which are used by different organizational subsystems in different levels of abstraction.

In the next three subsections we outline a user interface which allows users to define the subtypes and their organization, we show how the GPC lattice improves the performance of retrieval operations, and finally we show how GPC can be used in the design and development of information architecture.

OUTLINE OF A SCHEMA DEFINITION LANGUAGE

Without going into notational details, we describe the kinds of information that users must provide in the schema definition of their databases. We use a Pascal like syntax for type definitions in our examples. Let us assume that a user wants to define a max-entity type E with categorical properties C_j and attributes A_i.

(1) Users define categorical properties of the entity type E. Categorical properties are defined in terms of their elements. Elements which are not defined in terms of other elements are atoms.

Example:

```
Entity Type EMP;
Categorical properties:
  JOB = (SEC, ENG);
  SEX = (M, F);
```

SEC and ENG are atoms of JOB. M and F are atoms of SEX.

(2) Users define the attributes A_i of the entity type E.

Example:

```
Attributes:
  NAME: string;
  ID: integer;
  AGE: 18..75;
  SALARY: 5000.00 .. 200000.00;
  SPECIALTY: (MEC, ELEC, CIVIL);
  CALL#: 1 .. 1000;
  DEPENDENTS: 0 .. 20;
  TYPINGSP: 20 .. 100;
  FIELDASS: string;
  OFFICE#: 1 .. 200;
  SPOUSE: string;
  SUPERVISOR: string;
```

(3) Users determine the attributes which are applicable to each min-entity type. If there are m categorical properties and p atomic elements per each categorical property then there are p^m min-entity types. If both p and m are large numbers then it will be difficult to define attributes applicable to min-entity types. To simplify this process we define a *property-matrix*. The property-matrix of a max-entity type E has M + 1 dimensions. The first m dimensions correspond to m categorical properties. Each dimension is indexed by the atoms of a categorical property. The last dimension is indexed by the attribute names. Property matrix is of type Boolean. If the entry (c_1, c_2, .. , c_m, A_r), where c_j is an atom of C_j, is zero, then the attribute A_r is not applicable to instances of entity type corresponding to the element $< c_1, c_2, .. , c_m >$. The property matrix can be filled in several different ways. The exact syntax and method depends on the environment of the system and the applications. One approach is using a tabular syntax. The table will have m + n rows (m rows for categorical properties and n rows for attributes) and p^m columns. Table 1 shows the tabular representation of the property-matrix for our example. The tabular form may not be proper if m + n is a large number and categorical proper-

ties have many atoms. In this case a linear syntax can be used. The definition of the property-matrix for EMP, using this approach follows:

```
Prop_Mat_E (*, *, {NAME, ID, AGE,
  SALARY}) := 1;
Prop_Mat_E (ENG, MAL, {SPECIALTY,
  CALL#, DEPENDENTS}) := 1;
Prop_Mat_E (ENG, FML, {SPECIALTY,
  OFFICE#, SPOUSE}) := 1;
Prop_Mat_E (SEC, MAL, {TYPINGSP,
  FIELDASS, DEPENDENTS}) := 1;
Prop_Mat_E (SEC, FML, {TYPINGSP,
  SUPERVISOR, SPOUSE}) := 1;
```

(4) Users define the entity instances by specifying the elements of their categorical properties and values of applicable attributes.

In steps 1–3 the schema of the entity is defined and in step 4 its instances are given. The information provided in steps 1–3 is used by the system to construct the GPC lattice of the entity. The algorithm for this construction is straightforward. The system uses the property-matrix of step 3 to partition the attributes among the nodes of the lattice. The elements of categorical properties given in step 4 are used to determine the min-entity types of each instance.

EFFICIENT RETRIEVAL OPERATIONS

The lattice of generalization per category improves the performance of the frequent retrieval operations in the environment described at the beginning of this section in three different ways:

(1) The set of instances that may qualify as the desired objects are clustered one entity type. For example, all MALE-ENGINEERs are in one group and all FEMALE employees are in another group.

(2) There exists access paths from any entity type to its generalizations and specializations. Therefore, when a search is to be broaden or restricted there is a direct path to the relevant generalizations (broader categories) and specializations (restricted categories).

(3) The lattice of generalization allows a semantic and natural vertical partitioning of the attributes of general entity type among its subtypes. Those attributes of an instance e which are relevant to the fact that "e" is an employee appear as attributes of EMP, while the attributes which are relevant because e is an engineer appear as attributes of ENGINEER.

Table 1

Tabular representation of Property-matrix for entity type E

Categorical Properties / Attributes	Min-entity Types			
JOB	ENG	ENG	SEC	SEC
SEX	MAL	FML	MAL	FML
NAME	1	1	1	1
ID	1	1	1	1
AGE	1	1	1	1
SALARY	1	1	1	1
SPECIALTY	1	1		
CALL #	1			
DEPENDENTS	1		1	
TYPINGSP			1	1
FIELDASS			1	
OFFICE #		1		
SPOUSE		1		1
SUPERVISOR				1

There are two extreme approaches to storing the components of tuples describing an entity. One is to store the components of the tuples separately and assemble them when needed. The other extreme is to store all the components of a tuple together. Experience has shown that the first approach results in poor performance (Chamberlin, 1981). The extent in which the attributes of entities can be stored together is limited by the normalization theory (Maier, 1983) and the occurrence of inapplicable null values. In addition, as the number of attributes increases, the cost of retrieval increases, and the possibility of the retrieved attributes being used together decreases. Some researchers have addressed this problem and have invented methods to partition tuples *vertically*. The idea is to store those attributes of the entity types, which are retrieved together, in close affinity. This work has recently been extended by Navathe, Ceri, Wiederhold, and Dou (Navathe, et al., 1984). In (Navathe, et al., 1984), to decide which attributes are usually retrieved together the system monitors the retrieval operations, while the partitioning induced by generalization per category is based on the semantics of the data.

DESIGN AND DEVELOPMENT OF INFORMATION ARCHITECTURE

An important application of GPC is in the design and implementation of information architectures for organizations. Let us assume an organizational information requirement analysis (OIRA) (Dickson, Wetherbe, 1985)

has been carried out as part of the MIS planning process for a given organization. OIRA idntifies the underlying organizational subsystems and the major categories of information, which are organized in a (category of information by organizational subsystem) matrix. The entries in this matrix provide information about the different relationships between categories of information and organizational subsystems. This matrix is a high level conceptual aid for developing an overall information architecture. It is also a useful framework and starting point for the development of a concrete information architecture, within which applications are designed and implemented. In this section we are concerneed with this latter phase of the information architecture development.

Information categories developed in OIRA are general categories. Usually they have subcategories in potentially seeveral levels. As an example consider the STUDENTS category of a university (Vogel, Wetherbe, 1985). PROSPECTIVE, ENTERING, RETURNING, GRADUATING, and ALUMNI are possible subcategories of the STUDENTS category. In general, different organizational subsystems are concerned with different subcategories of information categories. For instance, in a university, Placement/Career Development subsystem is concerned with the GRADUATING and ALUMNI subcategories, while its Strategic Planning/Institutional Research subsystem is concerned with the RETURNING subcategory.

Although they are not considered in early stages of OIRA and are not included in category of information by organization subsystem matrix, each category of information and its subcategories have many attributes. Different organizational subsystems using the same category of information are usually concerned with a subset of these attributes. Again using the university example, the Instruction/Curriculum subsystem uses a different set of attributes of the STUDENTS category than the Legal/ Law Enforcement subsystem.

Finally, the information architecture must accommodate the information used in various levels of the organization, such as operations, management, control, and strategic planning. In each level different subsets of the attributes of the relevant categories and subcategories are used. The Strategic Planning/Institutional Research subsystem, for instance, is not concerned with detailed information (such as grades and telephone numbers) about all the RETURNING STUDENTS, although it is concerned with this subcategory in general.

GPC allows information to be organized according to categories and their subcategories, and in different levels of abstraction. By using GPC, the semantics of the data as it relates to the overall operation and objectives of the organization (discovered in the process of MIS planning) can be built into the information architecture. The following is an overview of the steps involved in using GPC for development of information architecture. We assume an OIRA has already been performed.

(1) Each category of information is treated as a categorical property and is defined as described in step 1 of section "Outline Of A Schema Definition Language".

(2) The attributes (characteristics) of categories and subcategories are defined.

(3) The property matrix is defined.

(4) The system forms the GPC lattice and organizes the attributes.

Step 1 takes the categories and their subcategories into account. Steps two and three take attributes of categories and their subcategories into account. The category of information by subsystem matrix can be extended and used to assign different nodes of the lattice to different subsystems. In general, managerial subsystems will be concerned with the nodes in the higher levels of the lattice and operational subsystems will use the lower level nodes.

Another application of GPC is in supporting different users' views of the data. The notion of external schema (view) in database technology is an attempt to allow different users to view relevant data with a desirable format. The problems with the view mechanism are: (i) views must be predefined by the users, (ii) vixews are supported by only some DBMS and in ad hoc manner, and (iii) update operation through view without violating integrity and consistency of the database is not possible in general (Cosmadakis, Papadimitriou, 1984). GPC is not a proposed replacement for the view mechanism, but it makes available to the users in different organizational levels, a large class of semantic views which are difficult to define and manipulate as external schemas.

Further Possibilities

In this section we briefly discuss some possible future enhancements of the theory and applications of GPC.

CONSTRAINTS

So far we assumed that all the nodes in the generalization lattice correspond to meaningful entity types. This is not always the case in real world applications. As an example if there is a constraint that requires all secretaries to be female then <SEC, M> will be an illegal entity type. Furthermore <{SEC, ENG}, M> and <SEC, {M, F}> entity types will degenerate to <ENG, M> and <SEC,

F>, respectively. A system based on GPC as described in this paper will create entity types corresponding to illegal combinations of categorical elements but these entity types will have no instances. The integrity subsystem of the system will be responsible to enforce these type of constraints and reject the instances with illegal combinations of categorical elements. The advantage of our system over conventional systems in this respect is that it requires no extra effort from the database designer to declare these constraints. The combinations of elements of categorical properties missing from the property matrix will be considered illegal. The system can enforce these integrity rules in the most efficient way because they are now built-in constraints.

ADDITION OF NEW CATEGORICAL PROPERTIES

Real world changes usually affect the databases which model the world. A relatively frequent change is addition of new categorical properties to the entities. As an example we may want to add MARITAL-STATUS = (MARRIED, SINGLE), or SENIORITY = (SENIOR, JUNIOR) to EMP. Mathematically, these extensions can be explained in terms of lattice product. Practically, the following questions are relevant:

(1) Do these extensions cause the reorganization of the whole database or not?

(2) What are the values of new categorical properties of the instances which already exist in the database?

We can not assume that the values of new categorical properties for old instances are unknown because this violates the assumption (ii) of section "The Lattice Of Generalization Per Category". A possible solution is to define default values for categorical properties and assume the old entities have these default values for their new properties.

APPROXIMATE CATEGORIZATION BASED ON PROPERTIES WITH CONTINUOUS VALUES

We have assumed that categorical properties are like enumeration data types and have discrete and limited number of elements. In practice some properties with continuous values are used for categorization of objects. When this is done user specfies the range of values for each element but this specification may be approximate. As an example, if we have AGE = (OLD, YOUNG) as a categorical property we may say approximately 50 or above is considered OLD. Currently, our system provides no help in partitioning of the continuous range of values to discrete elements. This must be done by users. It is a feasible and useful extension to provide data definition facilities for continuous categorical properties and data manipulation facilities to run fuzzy queries based on these properties.

Concluding Remarks

Generalization is a useful concept for organizing the information. It has been used in artificial intelligence as IS-A relationship, is supported in programming languages SIMULA and SMALLTALK (Goldberg, 1983) through class/subclass mechanism, and is a structural element of semantic data models in database.

We experienced difficulties when we applied the concept to the design of databases about entity types which can be generalized according to several properties. Suppose the entity type has **m** categorical properties (properties that are used for generalization). Further suppose each category has an average of **p** atoms (elements that are not generalization of other elements). There can be a maximum of 2^{p^m} entity types in the database. Even for small p and m it is infeasible to require the database designer to define all these entities.

We have solved this problem by developing a mathematical theory for the generalization per category. We have shown that:

(i) The generalizations of elements of a categorical property form a complete lattice.

(ii) Aggregation of several categorical properties form a complete lattice which is the direct product of the lattices of the categorical properties.

(iii) The generalizations of entity types per category form a lattice which is isomorphic to the lattice of the multiple categorical properties.

This formalism is the major theoretical contribution of our work.

The insight gained by this formalism allows the development of systems which can take more responsibilities in the management of data. Users define specialized entity types by using high level data definition languages and the system constructs the lattice of generalizations. The nodes of this system generated and controlled lattice are clusters of similar data objects. These clusters are semantically related to each other because they represent the same types of objects in different levels of abstraction. The edges are access paths from one cluster to others. The clusters and access paths ensure effective browsing of the database, efficient execution of frequent queries, and availability of semantic views of the data.

This is the major practical contribution of our work.

ACKNOWLEDGEMENTS

We thank S. March of MIS Research Center at the University of Minnesota and the ICIS anonymous referees for their careful review and constructive comments.

REFERENCES

Chamberlin, D.D., et al., "A History and Evaluation of System R," *CACM*, Vol. 24, No. 10, October 1981, pp. 632-646

Codd, E.F., "Extending The Database Relational Model To Capture More Meaning," *ACM TODS*, Vol. 4, No. 4, December 1979, pp. 397-434.

Cosmadakis, S.S., Papadimitriou, C.H., "Updates of Relational Views," *JACM*, Vol. 31, No. 4, October 1984, pp. 742-760.

Date, C.J., "Data Models", *An Introduction to Database Systems*, V. 2, pp. 210-230. Addison Wesley, 1983.

Davis, G.B., Olson, M.H., "Developing A Long-Range Information System Plan", *Management Information System*, McGraw-Hill 1985, Second Edition, pp. 443-472.

Dickson, G.W., Wetherbe, J.C., "Strategic Planning For MIS", *The Management of Information Systems*, McGraw-Hill, 1985, pp. 119-161.

Goldberg, A. and Robson, D. "Part One", *Smalltalk-80: The Language And Its Implementation*, Addison Wesley, 1983.

Hammer, M., McLeod, D., "Database Description With SDM: A Semantic Database Model", *ACM TODS*, Vol. 6, No. 3, September 1981, pp. 351-386.

Kent, W., "Limitations of Record-Based Information Models", *ACM TODS*, V. 4, N. 1, March 1979, pp. 107-131.

Liskov, B.H., Snyder, A., Atkinson, R., Schaffet, C. "Abstraction Mechanisms in CLU", *CACM*, 208, August, 1977, PP 564-576.

Maier, D., "Databases And Normal Forms", *Database Theory*, Computer Science Press, 1983 pp. 93-122.

Mylopoulos, J., Bernestein, P., Wong, H.K., "A Language Facility For Designing Database Intensive Applications", *ACM TODS*, Vol. 5, No. 2, June 1980, pp. 185-207.

Navathe, S., Ceri S., Wiederhold, G., Dou J., "Vertical Partitioning Algorithm for Database Design", *ACM TODS*, Vol. 5, No. 2, December 1984, pp. 680-710.

Sidle, T.W., "Weakness of Commercial Database Management Systems in Engineering Applications", *17th Proceedings Design Automation Conference*, June 1980.

Smith, J., Smith, D., "Database Abstractions: Aggregation", *CACM*, Vol. 20, No. 6, June 1977, pp. 405-413.

Smith, J., Smith, D., "Database Abstractions: Aggregations and Generalizations", *ACM TODS*, V. 2, No. 3, September 1977.

Tsichritzis, D.C., Lochovsky, F.H., "Data And Data Models", *Data Models*, Prentice-Hall, 1982 pp. 3-15.

Vassiliou, Y., "Null Values In Database Management: A Denotational Semantics Approach", *Proceedings ACM SIGMOD*, Boston, Mass. May 30-June 1, 1979.

Vogel, D.R., Wetherbe, J.C., "University Planning: Developing A Long-Range Information Architecture", MISRC-WP-85-08, Management Information Systems Research Center, University of Minnesota.

Wetherbe, J.C., Davis, G.B., "Developing a Long-Range Information Architecture", MISRC-WP-83-08, Management Information Systems Research Center, University of Minnesota.