INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of "sectioning" the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.

University Microfilms International
300 N. Zeeb Road
Ann Arbor, MI 48106

Baik, Ki Hoon

A TOPOLOGICAL APPROACH TO DATABASE THEORY

The University of Oklahoma PH.D. 1984

THE UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

A TOPOLOGICAL APPROACH TO DATABASE THEORY

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

degree of

DOCTOR OF PHILOSOPHY

BY

KI HOON BAIK

Norman, Oklahoma

1984

A TOPOLOGICAL APPROACH TO DATABASE THEORY

A DISSERTATION

APPROVED FOR THE

SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

BY

Leslie Miller

Charles R. McClure

A.R. Hurson

A. Frank

S. Lakshmivarahan

DISSERTATION COMMITTEE

ABSTRACT

The data modeling techniques which are currently used to construct the logical view of data have made use of trees, graphs, and sets, as underlying data structures. However, these structures have certain limitations when they are used in dynamic database representation for database design procedures. The topological approach to the data modeling makes use of abstract spaces and topologies as mathematical structures. The database space which serves as a logical view of the data is introduced with the concept of topological space. Then, the topological structure provides the concept of equivalence testing between database spaces. Furthermore, the concept of database space equivalence provides the foundation of the database generator which is an automatic database design machine. In the database space, the microcosmic set represents the set of attributes and it plays a major role for the intra-record type representation, while the selected topology represents the set of pseudo-entity types and its product topology plays a major role for the inter-record type representation in the database design level procedures.

ACKNOWLEDGEMENTS

The author wishes to express his appreciation to his advisor, Dr. Leslie L. Miller, for guidance, encouragement, and assistance throughout the course of research and the preparation of the manuscript. In addition, he is indebted much to his dissertation committee: Drs. S. Lakshmivarahan, Sudarshan Dhall, Ali Hurson, and Charles McClure for their suggestions and critical reading of the manuscript. The author gratefully acknowledges the School of Electrical Engineering and Computer Science which he has served as a Graduate Teaching Assistant throughout his graduate program.

This manuscript, except for some special characters and figures, was edited on VAX 11/780 and produced by a printer IBM 1403 on the IBM 3081 Model D at Merrick Computer Center, University of Oklahoma at Norman. The support facilities at the University of Oklahoma are appreciated.

My deepest appreciation must be expressed to my wife, Aekyung, for her patience and encouragement throughout the period of my overseas study. During the preparation of the manuscript, my first born son, Sanders, was born. A special appreciation is also intended to my son, Sanders, for his healthy growth.

TABLE OF CONTENTS

LIST OF DEFINITIONS

LIST OF FIGURES

A TOPOLOGICAL APPROACH TO DATABASE THEORY

CHAPTER I
INTRODUCTION

[1.1] FUNDAMENTAL APPROACHES

The proper representation of information can be used to aid human intelligence. This will continue to be one of the most important parts of computer systems. A database, which provides information, needs to be organized in a way that can be processed effectively. It must represent as closely as possible the real world situation, and be suitable for representation by computers. The range of data structures, currently used to support database organizations is a critical factor which affects system language, storage organization, and application programs. According to the underlying data structures, database systems are categorized into three fundamental approaches which are the hierarchical approach, based on trees, the network approach, based on graphs, and the relational approach, based on sets. They are described in Date[1981] and Tsichritzis-Lochovsky[1982].

The hierarchical approach and the network approach were developed from early file processing and report generator systems. They grew out of early attempts to implement integrated data systems. They are geared to the physical database with less concern for the logical database and the general database user population. Codd[1970] introduced the relational approach to strengthen the logical database and increase data independence. This is accomplished by introducing a powerful query language and using an axiomatic approach for deriving the database design procedures. In the early years of the relational model, there was a great debate concerning the relative merits of network and relational approaches, described in Rustin[1974] and Michaels-Mittman-Carlson[1976]. The conclusion was that there is no best approach such as there is no best programming language in any absolute sense.

The recent trend in database theory has been to concentrate on the relational approach. After examining the relational database model, one realizes that the relation is inadequate to store the complete semantics of the data. Relations do not provide the appropriate facilities for storing the structure of the relationships between data items. Chen[1976] and Codd[1979] have attempted to solve the problem by introducing the entity-relationship model and the extended relational model, respectively. In these efforts, database representation has been examined by a com-

bination of sets and graphs without much concern for new data structures such as abstract spaces, which support the database design procedures.

[1.2] PROBLEM DESCRIPTION

We believe the ultimate goal of database design theory is to build a database generator as an automatic database design machine. The first step on the way to the database generator is to establish the concept of equivalence between databases. For a given set of attributes, Beeri-Mendelzon-Sagiv-Ullman[1981] have taken an initial look at the concept of equivalence in database design. They have described an equivalence feature for dependency and representation in a fixed set of attributes. This is an interesting approach. However, we believe, in order to get to the database generator, the concept of database equivalence must be explored in general for classification of databases.

The problem description raised on the relational model can be demonstrated by a simple example. Let us consider relations, R1 and R2, given as follows:

R1: X1 = { a,b,c,d }, F1 = { {a}-->{b} },

R2: X2 = { p,q,u,v }, F2 = { {u}-->{v} },

where X1, X2 are sets of attributes and F1, F2 are sets of functional dependencies on X1, X2, respectively. Looking at

this example, we can observe a certain equivalence between the two relations. When R1 is decomposed into a third normal form, it will be {a,c,d}, {a,b} while R2 be {p,q,u}, {u,v}. If we introduce a bijection h, such as

$$h = \{\ a \rightarrow u,\ \ b \rightarrow v,\ \ c \rightarrow p,\ \ d \rightarrow q\ \}$$

which is a one-to-one onto mapping, then the two decompositions have exactly the same process. However, finding an appropriate bijection h is not easy task. Without regarding the dependencies, the number of possible bijections is 4!. In general, there are n! bijections if $n = |X1| = |X2|$.

One effective way of finding appropriate bijections is to replace the sets used in the relational model with topologies. Thus, we must take certain information contained in F1, F2 into X1, X2. This task is described as topologizing the set of attributes. One elegant way to pursue this task is to first examine the properties of F1, F2 and organize them to certain structures, then take the derived structures into X1, X2. By topologizing the set of attributes, it has certain structures which are topologies. This mathematical structure provides us with an effective means of generating the appropriate bijections. Thus, the topological approach is based on topologies as the underlying abstract space for the database design procedures.

[1.3] ORGANIZATION OF DISSERTATION

The basic concepts of topology and database theory are summarized in Chapter II. Then, Chapter III describes a topological view of data combining the concepts of topology and database. An ideal topology, called an optimal topology in this discussion, is derived from the axiomatization for dependencies. Chapter IV describes the concept of database equivalence by using the derived topological structures. Topics on the decomposition issues for the topological database model are covered in Chapter V. Finally, the conclusion and comments on future research for the topological approach are given in Chapter VI.

The main frame of the text consists of DEFINITIONs, THEOREMs, and EXAMPLEs. We have given 23 definitions which are vital seeds of the text. Then, they are supplemented by theorems and examples. Throughout the text, we try to use standard set theory notation. Finally, we like to familiarize the reader with our usage of square brackets. As an example, by BASE[T], we mean the name of the blackbox is BASE, the input of the blackbox is T, and the output of the blackbox is BASE[T].

CHAPTER II

PRELIMINARIES

This chapter contains all of the fundamental concepts of topology and database theory which are necessary for introducing the dissertation work. Terminology and notation selected from topology and database theory are reviewed briefly. An effort has been made to standardize the notation. Certain notation in database theory has been replaced with standard set theory notation.

Section[2.1] describes the topology concepts that have been used in the development of the topological approach. Topology is one of oldest topics in mathematics. Plenty of references are available to the interested reader, such as Bourbaki[1948], Dugundji[1966], and Cullen[1968] which are introductory textbooks. Also, Steen-Seebach[1978] provides elegant examples on various topologies. Hu[1965] gives an advanced picture of theory in topology.

Section[2.2] summarizes the fundamentals of database theory used in the dissertation work. Database theory is one of vital new topics in computer science. Date[1981], Tsichritzis-Lochovsky[1982], and Ullman[1982] have written

introductory textbooks that will provide the interested reader additional background in the general topic of database models. In addition, Maier[1983] describes the relational approach by combining much of the recent work on the relational database model.

[2.1] TOPOLOGY

A mathematical model is a set with certain structures defined on it. A structure defined on a set is called a topological structure. Let us begin with the definition of the topological space. This space will give us the starting point for the theoretical approach in the discussion of the dissertation work. Since we only treat nontrivial finite systems in computer science, we mean a non-empty finite set by simply saying a set throughout this discussion unless we specify otherwise.

DEFINITION[2.1]. TOPOLOGICAL SPACE.

A topological space is defined as a pair (X,T) consisting of a set X and a collection of subsets of X, satisfying the axioms: A1, A2, and A3, where

A1: $\emptyset$, X $\in$ T,

A2: A, B $\in$ T ==> A$\cap$B $\in$ T,

A3: A, B $\in$ T ==> A$\cup$B $\in$ T. <>

The collection T is called a topology on X. Elements of T are called open sets. Elements of X are called points. A topological space (X,T) is sometimes referred to as a space X when the topology T is obvious. Topologizing a set is the general process of introducing a topology on the set. The most popular process for topologizing a set X is to start from the arbitrary family W of subsets of X. The set W leads to a unique topology containing W by using the concepts of bases and subbases which are defined as follows:

DEFINITION[2.2]. BASE[T].

A base for a topology T, denoted BASE[T] is a subset of T such that every element of T is a union of elements of the BASE[T]. <>

DEFINITION[2.3]. SUBBASE[T].

A subbase for a topology T, denoted SUBBASE[T] is a subset of T such that the base for T consists of intersections of elements of the SUBBASE[T]. <>

Note that from the definitions, there are quite a few bases and subbases for a topology. The smallest base in size is called a minimal base and the smallest subbase a minimal subbase. For a multiple intersection or union, the collection of operands is called an operand set. Note that operand sets must be finite but can be empty.

EXAMPLE[2.1]. Consider a set: X = {a,b,c,d,e} and

a subbase: Ts = { {a,b}, {b,d} }.

Then, we get a base: Tb = { X, {a,b}, {b}, {b,d} } and

a topology: T = { $\emptyset$, X, {a,b}, {b}, {b,d}, {a,b,d} }.

In this case, observe that Ts is a minimal subbase and Tb is a minimal base. <>

THEOREM[2.1]. For an arbitrary family W of subsets of X, there always exists a unique minimal topology in size containing W. <>

PROOF: Consider W = SUBBASE[T]. Then, SUBBASE[T] forms BASE[T] by taking all possible intersections of elements and BASE[T] forms topology T by taking all possible unions of elements. Operations, union and intersection are uniquely determined. <>

Let Td denote the class of all subsets of X. Observe that Td satisfies the axioms for a topology on X. Td is called the discrete topology. The notation Td is used for the discrete topology throughout this discussion. The class {$\emptyset$,X} consisting of $\emptyset$ and X alone is itself a topology on X which is called the indiscrete topology.

DEFINITION[2.4]. PARTITION TOPOLOGY.

A partition topology is a collection of subsets of a set X satisfying the axioms: A1, A2, A3, and A4, where

A4: $A \in T \Rightarrow \overline{A} \in T$. <>

A partition topology is denoted by $\tilde{T}$ throughout this discussion. Note that a topology is a closed collection under set operations: intersection and union and that a partition topology is a closed collection under set operations: intersecton, union, and complement. Non-empty sets P and Q are called a decomposition of X, if $P \cup Q = X$ and

called a partition of X, if $P \cup Q = X$ and $P \cap Q = \emptyset$.

Note the partition is a special case of decomposition. A set can be represented by a set of its subsets through the notion of decomposition.

Let T1 and T2 be topologies on a set X. Suppose that $T1 \subseteq T2$. We say that T1 is coarser than T2 or that T2 is finer than T1. Observe the collection of all topologies on X is partially ordered by the inclusion property. The discrete topology is the finest topology and the indiscrete topology is the coarsest topology. All other topologies reside between them. Thus, the bound for sizes of topologies on X is given as follows:

$$2 \leq |T| \leq 2^n, \quad \text{for} \quad n = |X|.$$

THEOREM[2.2]. Let T1 and T2 be topologies on a set X. Then, $T1 \cap T2$ is always a topology on X. <>

PROOF: For arbitrary $A, B \in T1 \cap T2$, observe $A \cap B \in T1$ and $A \cap B \in T2$. Then, $A \cap B \in T1 \cap T2$.

The same argument on the union operation will give the desired result. <>

The union of topologies on the same set is not a topology in general. However, we can observe the maximum size of topology constructed by the union of two topologies.

THEOREM[2.3]. Let (X,T1), (X,T2) be topological spaces. Considering a topology T such that $T1 \cup T2 = SUBBASE[T]$, we can get the following condition:

$$|T| \leq 2|T1||T2| - 3|T1| - 3|T2| + 6. \quad <>$$

PROOF: Observe that new elements of T come from intersections and unions between elements of T1 and T2. Considering the set of new elements, $T = D \cup T1 \cup T2$, where

$$D = \{ P, Q \mid P = Ai \cap Bi, \quad Q = Ai \cup Bi,$$
$$Ai \in (T1 - \{\emptyset, X\}), \quad Bi \in (T2 - \{\emptyset, X\}) \}.$$

Since $|D| \leq 2(|T1|-2)(|T2|-2)$, we get

$$|T| \leq 2(|T1|-2)(|T2|-2) + |T1| + |T2| - 2.$$

This gives the desired result. <>

Let (X1,T1), (X2,T2) be topological spaces. A function f from X1 to X2 is continuous if and only if $P \in T2$ implies $f^{-1}[P] \in T1$. The bijection f is called a homeomorphism if f and f^{-1} are continuous. The notation f^{-1} is used for the inverse of the function f.

DEFINITION[2.5]. TOPOLOGICAL EQUIVALENCE.

Let (X1,T1), (X2,T2) be topological spaces. Then, (X1,T1) and (X2,T2) are topologically equivalent if and only if there exists a homeomorphism f: X1 --> X2. <>

EXAMPLE[2.2]. Consider the topological spaces, (X1,T1) and (X2,T2) as follows:

X1 = {a,b,c,d}, T1 = { $\emptyset$, X1, {a}, {a,b}, {a,b,c} },

X2 = {p,q,u,v}, T2 = { $\emptyset$, X2, {p}, {q}, {p,q}, {q,u,v} }.

Also consider the functions, f: X1 --> X2, g: X1 --> X2:

f = {(a,q),(b,u),(c,v),(d,u)},

g = {(a,p),(b,p),(c,u),(d,v)}.

The function f is continuous since the inverse of each member of the topology T2 on X2 is a member of the topology T1 on X1. The function g is not continuous, since {q,u,v} is in T2 but its inverse {c,d} is not in T1. <>

Consider a topological space (X,T). Let p be a point on (X,T). A subset N of X is called a neighborhood of p if N is a superset of an open set containing p. The class of all neighborhoods of a point p in X is called the neighborhood system of the point p.

Let Y be a subset of X. PROJECT[T:Y] is defined as the class of all intersections of Y with open sets in T. Observe that PROJECT[T:Y] is a topology on Y. A topological space (Y,PROJECT[T:Y]) is called a subspace of (X,T) and PROJECT[T:Y] is called a relative topology of the topology T on Y. Operator PROJECT has more general meaning in the next section. In general, PROJECT[X:A] is the collection of intersection with A or A-component of elements in the set X.

DEFINITION[2.6]. PRODUCT TOPOLOGY.

The product topology for a topology T, denoted T^2 is defined as the collection of all ordered pairs (A,B), where A, B $\in$ T. <>

There is an alternate definition for the product topology in topology literature. The product topology T1XT2 for topologies, T1 and T2, is defined in the literature as the collection of AXB such that A $\in$ T1 and B $\in$ T2. However in this discussion, we use the notation (A,B) instead of AXB. The advantage of this notation will become clear as we explore the topological database model in the next chapter.

EXAMPLE[2.3]. Consider a topological space (X,T):

X = {a,b,c,d,e},

T = { $\emptyset$, X, {a,b}, {b}, {b,d}, {a,b,d} }.

Also consider a subset of X: Y = {b,c,d}.

Then, a subspace (Y,Ty):

Y = {b,c,d},

Ty = PROJECT[T:Y] = { $\emptyset$, Y, {b}, {b,d} }.

The product topology for Ty:

Ty^2 = { ($\emptyset$,$\emptyset$), ($\emptyset$,Y), ($\emptyset$,{b}), ($\emptyset$,{b,d}), (Y,$\emptyset$), (Y,Y), (Y,{b}), (Y,{b,d}), ({b},$\emptyset$), ({b},Y), ({b},{b}), ({b},{b,d}), ({b,d},$\emptyset$), ({b,d},Y), ({b,d},{b}), ({b,d},{b,d}) }. <>

A parity topology, denoted <T> throughout this discussion, is a topology such that each element of an open set has a parity as an intrinsic property. Paritizing a topology is the general process for introducing parity in the topology. This process can be introduced by defining certain paritizing operators. If there is no policy on the paritizing, all possible combinations of parity on open set are considered. An actual example for a paritizing operator will be shown in the next chapter.

A relationship between topological and graphical structures can be expressed by a base graph for a given topology. A minimax intersection is a non-empty intersection with the restriction on its operand set such that any proper subset of the operand set except singleton sets is not allowed as an operand set and any element of the operand set is not a proper subset of the others.

DEFINITION[2.7]. BASE GRAPH FOR TOPOLOGY.

Let (X,T) be a topological space. A base graph for the topology T is defined as a pair (V,E), where

V: a set of vertices which is the collection of minimax intersections for the elements of minimal base for the topology T, including X and excluding $\emptyset$,

E: a set of edges which is the collection of partial orders of inclusion properties for the vertex set, excluding transitive orders. <>

EXAMPLE[2.4]. Consider a topological space (X,T):

X = {a,b,c,d,e,f},

SUBBASE[T] = {{a,b,c},{a,b,d},{a,e},{b,f}}.

Then, a minimal base for T:

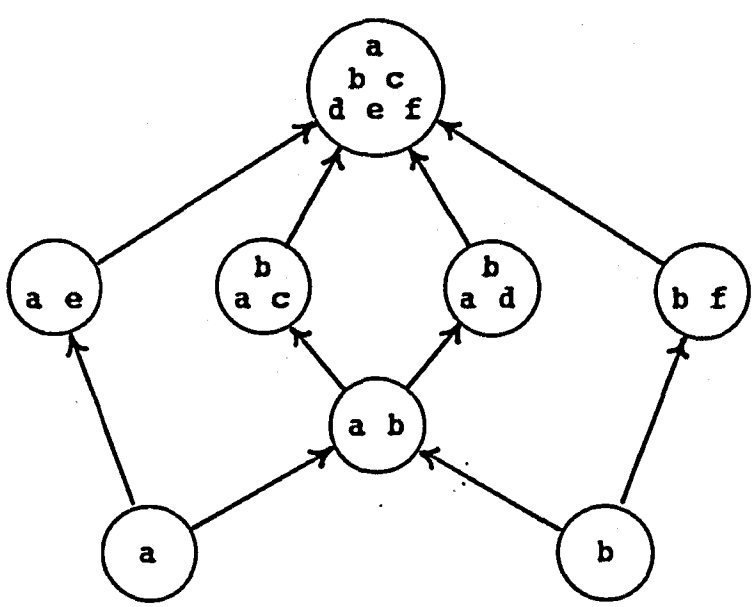
{ {a,b,c}, {a,b,d}, {a,e}, {b,f}, {a}, {b} }.

The base graph for T is (V,E), where

V = [X, {a,b,c}, {a,b,d}, {a,e}, {b,f}, {a}, {b},
{a,b}],

E = [({a,b,c},X), ({a,b,d},X), ({a,e},X), ({b,f},X),
({a,b},{a,b,c}), ({a,b},{a,b,d}), ({a},{a,e}),
({b},{b,f}), ({a},{a,b}), ({b},{a,b})],

as given in FIGURE[2.1]. <>



FIGURE[2.1]. An example of a base graph.

THEOREM[2.4]. Let (X,T) be a topological space and (V,E) be the base graph for T. Then, $|V| < 2|X|$. <>

PROOF: A set V consists of a minimal base for T and new vertices created by minimax intersections. Since each element of the minimal base must have a representative element of X, the maximum size of the minimal base for T is $|X|$. Then, for the minimal base, consider the collection of operand sets for minimax intersections on the elements of the minimal base. Since each element of this collection must have a representative element of the minimal base, the maximum number of creations by minimax intersections is less than $|X|$ which is the maximum size of the minimal base. This gives the desired result. <>

The size of base graph has linear bound. This is an important result because the complexity of an algorithm to handle a base graph is a function of its size. From the set inclusion properties, observe that a base graph is an acyclic digraph with a sink. The interested reader is directed to Harary[1969] for details in introductory graph theory. The base graph for topology is uniquely determined since minimal base and the minimax intersections of its elements are unique. The concept of the base graph will be used to find appropriate homeomorphisms in Chapter IV. This is a brief review of the relevant topics in topology.

[2.2] DATABASE THEORY

The first standardization of database systems was the report from CODASYL DBTG[1971] which is based on graph structures. Thus, the CODASYL terms are briefly examined at the beginning of this discussion. In the CODASYL approach, a view of the concept of a record requires three levels of abstraction: a record structure, an occurrence of the record structure with an entity, and an instance of the record with values assigned to each of the record fields. CODASYL refers to the record structure as a record type, the record occurrence associated with a particular entity as a record, and the set of values assigned as an instance of the record. Records and fields are used in terms of conventional information processing. A field can hold one unit of information and a record represents an entity by using a collection of fields. In this discussion, notation t[X] is used for a record where X is a record structure. If $Y \subseteq X$, then Y is called a pseudo-entity type and t[Y] is called Y-component of the record t. Records provide an excellent tool for processing information, described in Hanson[1982]. Also, Kent[1979] describes the basic assumptions behind the record based information processing.

The CODASYL approach allows a one-to-many relationship, called an ownership in this discussion and a set in the original DBTG report, between records. An ownership contains

two kinds of records: an owner record of which there is exactly one and any number of member records. The CODASYL approach defines an ownership type as a named relationship between record types. The ownership type consists of exactly one owner record type and a number of member record types. Thus, the concept of ownership requires three levels of abstraction: an ownership structure, an occurrence of the record structure with a named relationship, and an instance of the ownership, described in Olle[1978]. The CODASYL approach refers to them as an ownership type, an ownership, and an instance of ownership, respectively. There is an ownership whenever there is an owner record. An ownership is said to empty when there are no member records. It is interesting to compare inter-record type with intra-record type. The inter-record is an ownership while the intra-record is a record. The choice of these two types for the database representation is one of the major roles of designer in the CODASYL approach.

The relational approach was introduced by Codd[1970] on the basis of the set theoretic approach. The relational approach determines the inter-record types dynamically while the CODASYL approach does so statically. There are no differences between the inter-record types and the intra-record types in the relational approach. In the relational approach, the main concepts are described with the intra-record types and the dependencies which are essentially the

same as the inter-record types in the CODASYL approach, as described in Tsichritzis-Lochovsky[1982].

A relation scheme is a set of attributes while a relation is a set of records. An attribute is specified by a domain which is a set of possible information values. Records of a relation are expressed as n-tuples if the associated relation scheme has n attributes. A functional dependency is expressed as a pair (X,Y) where X and Y are sets of attributes. The notation X --> Y is replaced in this discussion with (X,Y) to allow the direct application of the product topology in the next chapter. A relation R satisfies a functional dependency (X,Y) if whenever there are two records s and t in R such that s[X] = t[X], it must also be true that s[Y] = t[Y]. A multivalued dependency is expressed as a pair (X,Y) where X and Y are sets of attributes. Again the notation X ->-> Y is replaced with (X,Y) to allow the application of the product topology. A relation R satisfies a multivalued dependency (X,Y) if whenever there are two records s and t in R such that s[X] = t[X], there must also exist a record w in R such that

$$w[X \cup Y] = s[X \cup Y] \quad \text{and} \quad w[\overline{X \cup Y}] = t[\overline{X \cup Y}].$$

Two operators, PROJECT and JOIN are introduced on a set of relations. PROJECT[R:X] is defined as a collection of the X-component of tuples in the relation R, where X is a subset of the relation scheme associated to the relation R.

JOIN[R1:R2:...:Rn] is defined as a collection of tuples t such that t[Xi] ∈ Ri, i = 1,2,...,n, where Xi is a relation scheme associated to the relation Ri. JOIN operator means a natural join in database literature. A join dependency is expressed as a collection of attribute sets. A relation R satisfies a join dependency { X1,X2,...,Xn }, if

X = X1 ∪ X2 ∪ ... ∪ Xn and

R = JOIN[PROJECT[R:X1]:PROJECT[R:X2]:....:PROJECT[R:Xn]],

where X is the relation scheme for the relation R.

In addition, we have the concept of embedded dependencies. Recall the relation scheme is a set of attributes. An embedded dependency is satisfied only in a subset of a relation scheme, but it is not satisfied for entire relation scheme. There exist embedded multivalued and embedded join dependencies. Throughout this discussion, we will use multivalued dependency and join dependency terminology, excluding embedded dependencies.

The concept of normal forms is introduced from the functional, multivalued, and join dependencies. The normalization process has been proposed as a decomposition theory. The main reason for the normalization is to remove the anomalies in the case of data insertion, deletion, and update. Two plausable conditions are known as lossless join property and dependency preservation property in the decomposition processes. In general, the original relation

should be recoverable and the data structuring dependencies should be preserved after the decomposition.

The closure set for a set of dependencies is the collection of dependencies logically implied by the set of dependencies. The notation F+ is used for the closure logically implied by the set F of functional dependencies. Let F be a subset of G, where F and G are sets of dependencies. Then, F is called a cover of G if all dependencies in G are logically implied by F. Let X be a relation scheme and K be a subset of X. Then, K is called a key if $(K,X) \in F+$ and $(N,X) \notin F+$ for $N \subset K$.

The inference problem in database theory is how to decide whether a dependency is logically implied by the given set of dependencies. The inference problem has been solved by the chase process, described in Maier-Mendelzon-Sagiv[1979]. When the chase algorithm is applied to the inference problem, we say the chase is successful if a dependency to be tested can be derived from the given set of dependencies.

Since the chase process is used to test the axiomatization for join dependencies in the next chapter, we examine it some detail. A successful chase process for join dependencies can be expressed as a acyclic digraph, as given in Sciore[1982]. Recall that a digraph consists of vertices

and directed edges. An acyclic digraph contains no directed cycles. A sink is a vertex which can be reached by all others. For a directed edge (Ri,Rj), Ri is adjacent to Rj and Rj is adjacent from Ri. A vertex is a n-tuple if the relation scheme has n attributes. Each component of a n-tuple is associated to each attribute with even or odd variable. For a set G of edges such that

G = { (Di,X) | Di ∈ D, D: a vertex set, X: a vertex },

D is associated to a join dependency and X is associated to a derived tuple. Initially, a set of vertices is given by a join dependency to be tested. A vertex associates a member of join dependency. A component of vertex has even variable if the member of join dependency associated with the vertex has the attribute associated with the component, odd variable otherwise. Even variables are uniquely assigned to each attribute while odd variables are assigned one for each. If the relation scheme has n attributes and the join dependency tested has k members, then there are n even variables and at most n(k-1) odd variables. A derived vertex is constructed in the way that for an attribute Xi in the member X of applied join dependency, Xi-component of derived tuple is the same as Xi-component of the vertex associated to the member X. Finally, the sink has all even variables.

EXAMPLE[2.5]. Consider a set U of attributes:

U = {a,b,c,d,e} and a set W of join dependencies:

M = {J1,J2,J3,J4}, where

J1 = {{a,b,c}, {c,d,e}}, J2 = {{a,d}, {a,e}, {b,c,e}},

J3 = {{b,d}, {a,b,c,e}}, J4 = {{a,b,e}, {a,c,d}},

and a join dependency which we want to derive:

J8 = {{a,b,c}, {b,d}, {b,e}, {a,e}, {a,c}}.

Note that notations J5, J6, and J7 are reserved for intermediate join dependencies implicitly. Then, we get a set V of vertices given initially as follows:

V = {V1,V2,V3,V4,V5}, where

V1 = (a1, a2, a3, b1, b2),

V2 = (b3, a2, b4, a4, b5),

V3 = (b6, a2, b7, b8, a5),

V4 = (a1, b9, b10, b11, a5),

V5 = (a1, b12, a3, b13, b14).

Note that a1,a2,...,a5 are even variable and b1,b2,...,b14 are odd variable. We get a set D of vertices derived by V:

D = {D1,D2,D3,D4}, where

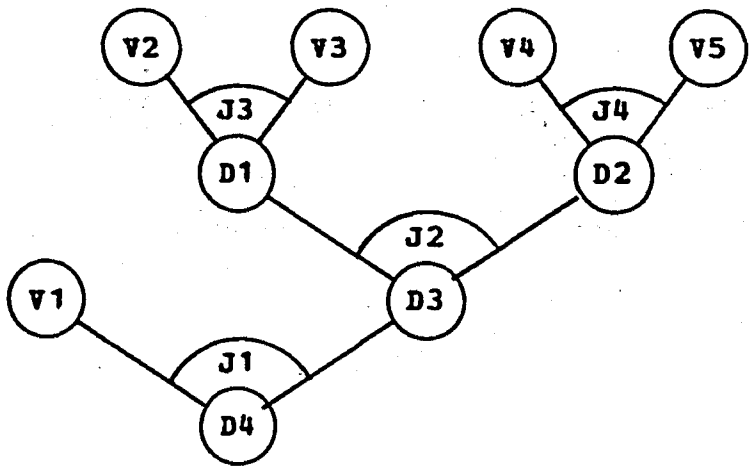
D1 = (b6, a2, b7, a4, a5),

D2 = (a1, b9, a3, b13, a5),

D3 = (b6, b9, a3, a4, a5),

D4 = (a1, a2, a3, a4, a5).

D1 is derived from V2, V3 by using J3, D2 from V4, V5 by J4, D3 from D1, D1, D2 by J2, and finally D4 from V1, D3 by J1. Note D4 is the sink for the successful chase process. The derivation graph is given in FIGURE[2.2]. <>



FIGURE[2.2]. A successful chase process.

Inference rules are a set of rules by which new dependencies can be derived from the given set of dependencies. A set of inference rules is complete for a set of dependencies if any dependency which is implied by the given set of dependencies can be derived from the given set of dependencies by using those inference rules. The complete axiomatizations for inference rules on functional, multivalued, and join dependencies have been explored by Armstrong[1974], Beeri-Fagin- Howard[1977], and Sciore[1982], respectively. Issues on the complete axiomatization for dependencies will be examined in great detail in the next chapter. This represents a brief review of the relevant topics in database theory. The interested reader can look at the cited references for more detail on a particular subject.

CHAPTER III

TOPOLOGICAL APPROACH

The primary purpose of data modeling techniques is to provide a representation for information and manipulation operators for such a representation. A database representation based on the topological view is the primary concern of this chapter. The database unit for the database representation in the topological model is an abstract space, called a database space. A database space is characterized by the corresponding topological space. Appropriately selected topologies provide the facility to store the semantics of the data constraints such as functional, multivalued, and join dependencies.

The definition of the database space is introduced in Section[3.1] by assuming an arbitrary topology. In the following sections, we will treat issues surrounding the selection of the topology. The obvious method for the selecting the topology is to base the topology on the data semantics of the application such as the data constraints. The properties of functional, multivalued, and join dependencies are examined and the complete axiom system of the relational

model are extended to the topological space by assuming the discrete topologies in Section[3.2]. They can be generalized in the case of arbitrary topologies.

In Section[3.3], we will examine the topologizing process, required to select an optimal topology. Section[3.4] introduces structures based on the optimal topology to axiomatize the concepts of functional, multivalued, and join dependencies. A series of algorithms to compute the structures are included.

[3.1] DATABASE SPACE

The database representation in the topological model is given by an abstract space, called a database space. Let us begin with the definition of the database space. Originally the database space was defined by Baik-Miller[1983] through the concept of the topological space as follows:

DEFINITION[3.1]. DATABASE SPACE (DBSP).

A n-dimensional Database Space, usually denoted by D^n is a n-dimensional space constructed by the n attributes as the n coordinates such that the set of n coordinates is a topological space. <>

Note that the most important portion of database space is the topology in the sense of storing data semantics. Since we have not explored the appropriate topology, we assume an arbitrary topology in this discussion.

A point in the n-dimensional DBSP, expressed by a n-tuple, reflects a record for an entity. In any instance, if data for a record is available then the point associated with the record is a valid point, otherwise it is an invalid point. A surface in the DBSP reflects a record for which information is partially available. A surface is a record with null values. Thus, the number of null values in the record is the degree of freedom of the surface. The set of all valid points including surfaces in the database space represents available information for the current instance of the database.

CITY	STATE	POPULATION
Norman	OK	71500
Miami	OK	14200
Miami	TX	800
Austin	TX	NULL

FIGURE[3.1]. A sample data set I.

EXAMPLE[3.1]. Consider a set X of attributes:

X = { CITY, STATE, POPULATION }.

A 3-dimensional DBSP is formed by the 3 coordinates:

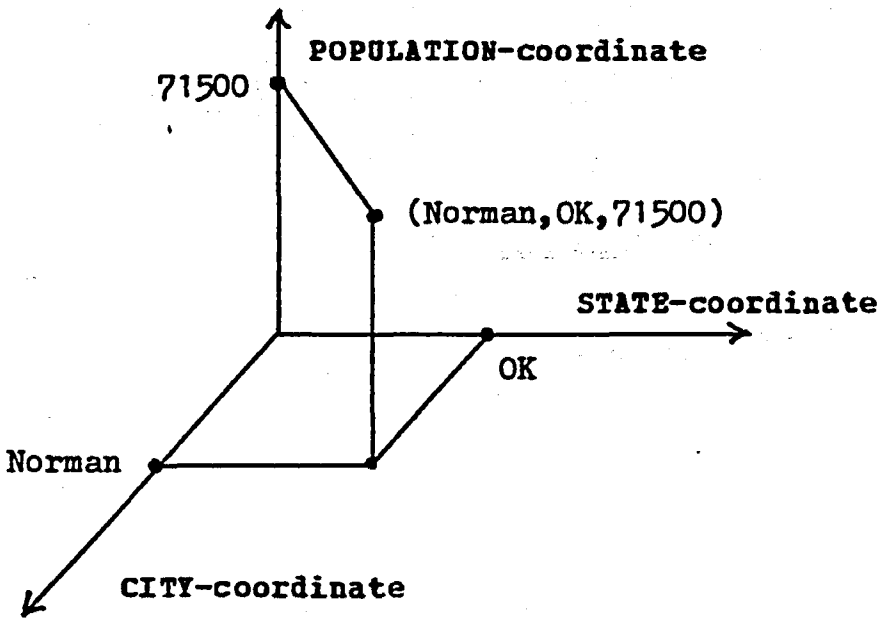
CITY, STATE, and POPULATION.

The CITY-coordinate represents a domain of the attribute CITY, a set of cities. The STATE-coordinate represents a domain of the attribute STATE, a set of states. The POPULATION-coordinate represents a domain of the attribute POPULATION, a set of positive integers. If we introduce an arbitrary topology T:

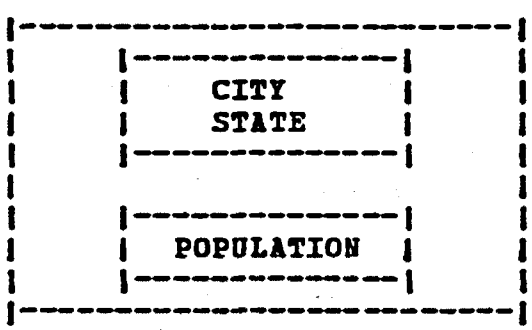
T = { $\emptyset$, X, {CITY,STATE}, {POPULATION} },

then (X,T) is a topological space. In a certain instance, if the information of three cities which are Norman, OK with population 71500, Miami, OK with population 14200, and Miami, TX with population 800, as given in FIGURE[3.1], is available, then three points in the DBSP, (Norman,OK,71500), (Miami,OK,14200), and (Miami,TX,800) are valid points in the instance. In the 3-dimensional case, the DBSP is expressed by the Euclidean Space, as given in FIGURE[3.2]. In another instance, if we know a city, Austin, TX with unknown population, then it is represented as a surface, (Austin,TX,NULL) which has the first degree of freedom. Thus, the surface, (Austin,TX,NULL) will be a straight line in the conventional Euclidean Space. <>

DIMENSION OF DATABASE SPACE



TOPOLOGY OF DATABASE SPACE



INSTANCE OF DATABASE SPACE

(Norman,OK,71500)
(Miami,OK,14200)
(Miami,TX,800)
(Austin,TX,NULL)

FIGURE[3.2]. An example of a Database Space.

Ignoring the dimensionality of database space, the database space is simply denoted by D without specifying its dimensionality. For a database space D, the dimension of D, denoted DIM[D] is the set of coordinates, the topology of D, denoted TOP[D] is a topology on DIM[D], and an instance of D, denoted INS[D] is the set of valid surfaces and points. Then, a database space D can be specified by DIM[D], TOP[D], and INS[D]. The topolgical space for the database space D is given as pair (DIM[D],TOP[D]). Sometimes when the instance of database space is ignored, a database space D is simply expressed as a topological space (X,T), where X = DIM[D] and T = TOP[D]. We will use both notations (X,T) and (DIM[D],TOP[D]), throughout this discussion.

We need to pay attention on the dimension of a database space. The dimension, DIM[D] is constructed by the set of coordinates. A coordinate is defined by the domain of an attribute. The domain of an attribute ai, denoted DOM[ai] is a set of values. Then, the size of a database space is determined by the size of its domains. By assuming

DIM[D] = {a1,a2,a3,...an} for a database space D,

the size of D:

$$|D| = |DOM[a1]| \times |DOM[a2]| \times \ldots \times |DOM[an]|,$$

where |DOM[ai]| is the size of DOM[ai].

Note that DOM[ai] must be a finite set, since it must be represented by computers. The semantics of the domains is determined by query language for users. Certain domains can

be the same set or a domain can be a subset of another domain. The information processing capability of database space is limited by determining the coordinates and their proper domains.

[3.2] AXIOMATIZATION FOR DEPENDENCIES

The best topology for our purpose can be selected by examining the data semantics of the application. The data constraints imposed on the database by the functional, multivalued, and join dependencies are the place to initiate our discussion. The dependencies are defined in terms of an abstract space using the discrete topology in this section. Axiomatization process for a set of functional dependencies was first explored by Armstrong[1974]. By assuming a discrete topology Td, Armstrong axiom system for a set of functional dependencies on Td, can be given as follows:

THEOREM[3.1]. A complete axiom system for functional dependencies on a discrete topology Td is given as follows:

F1: $Y \subseteq X \implies (X,Y) \in F[Td]$,

F2: $(X,Y) \in F[Td] \implies (X \cup Z, Y \cup Z) \in F[Td]$,

F3: $(X,Y), (Y,Z) \in F[Td] \implies (X,Z) \in F[Td]$,

where $X, Y, Z \in Td$,

F[Td]: a set of functional dependencies on Td. <>

PROOF: See APPENDIX[A.1]. <>

Note the collection, F[Td] is a subset of the product topology Td^2 of the discrete topology Td. Also, F[Td] is a closed collection on Td, introduced by using the complete axiom system. In the case of an arbitrary topology, the closed collection F[T] for a topology T can be generalized, because if $X, Y, Z \in T$ then $X \cup Z$, $Y \cup Z \in T$.

The advantages of the generalization to an arbitrary topology will become clear when we are to select an optimal topology in the next section. In the case of the discrete topology, F[Td] = F+ where F+ is the closure for the set F of functional dependencies. Axiomatization process for a set of multivalued dependencies was explored in the paper given by Beeri-Fagin-Howard[1977]. By assuming a discrete topology Td, a complete axiom system for a set of multivalued dependencies on $\widetilde{Td}$ can be given as follows:

THEOREM[3.2]. A complete axiom system for multivalued dependencies on a discrete topology Td is given as follows:

M1: $(X,Y) \in F[Td] \Rightarrow (X,Y) \in M[Td]$,

M2: $(X,Y) \in M[Td]$, $(X \cup Y, Z) \in F[Td]$
$\Rightarrow (X, \overline{Y} \cap Z) \in F[Td]$,

M3: $(X,Y) \in M[Td] \Rightarrow (X, \overline{X \cup Y}) \in M[Td]$,

M4: $(X,Y) \in M[Td] \Rightarrow (X \cup Z, Y \cup Z) \in M[Td]$,

M5: $(X,Y), (Y,Z) \in M[Td] \Rightarrow (X, \overline{Y} \cap Z) \in M[Td]$,

where $X, Y, Z \in \widetilde{Td}$,

M[Td]: a set of multivalued dependencies on $\widetilde{Td}$. <>

PROOF: See APPENDIX[A.2]. <>

Axioms, M1 and M2 state that functional dependencies are the special case of multivalued dependencies by their definitions. Note that M[Td] is a subset of the product topology Td^2 of the discrete topology Td. Also, M[Td] is a closed collection on $\tilde{Td}$, introduced by using the complete axiom system for multivalued dependencies. In the case of an arbitrary partition topology, the closed collection M[T] for a topology T can be generalized, because

if $X, Y, Z \in \tilde{T}$ then $\overline{X \cup Y}$, $Y \cup Z$, $\overline{Y} \cap Z \in \tilde{T}$.

Since axioms M3 and M5 involve a complement operation, this axiom system can be generalized to an arbitrary partition topology which is a closed collection under the complement operation. Again here, the advantages of the generalization to an arbitrary topology will become clear when we select an optimal topology in the next section. In the case of the discrete topology, $Td = \tilde{Td}$ and $M[Td] = M^+$ where M^+ is the closure for the set M of multivalued dependencies.

The axiomatization process for a set of generalized join dependencies was examined by Sciore[1982]. However, in this discussion, by introducing the concept of parity topology, the complete axiom system is constructed and a closed collection G[Td] on a parity discrete topology <Td> is derived. Since the complete axiom system is given on a parity discrete topology, we need to define paritizing process. Consider three functions given as follows:

EVEN[X] = a set of even members of a set X,

ODD[X] = a set of odd members of a set X, and

ROOT[X] = a set X without parities.

Now, we can define a paritizing operator as follows:

PARITY[A] = { Di | A = { X | X = ROOT[Di] },

ROOT[EVEN[Di]]∩ROOT[ODD[Dj]] = $\not\emptyset$ },

where Di, Dj are elements of <Td> and an operand A is a set of open sets in a discrete topology. One can see from the definition that an element of the sets of operand A will pick up either an even or odd parity and retain it throughout the paritizing process. We use notation a+ to denote even parity and a- to denote odd parity for an element a in this discussion.

EXAMPLE[3.2]. Consider a set A: A = { A1, A2, A3 },

where A1 = {a,d}, A2 = {a,e}, A3 = {b,c,e}.

Then, an example of the application of PARITY is:

PARITY[A] = { D1, D2, D3 }, where

D1 = {a-,d+}, D2 = {a-,e+}, D3 = {b-,c+,e+}.

Also, examples of introduced functions are:

ROOT[D1] = A1, EVEN[D1] = {d+}, ODD[D1] = {a-}. <>

Observe the operator PARITY determines the multiple results of the operation. All possible results should be considered. The deparitizing process is to simply remove their parities. We have examined the preliminaries for introducing a complete axiom system for join dependencies on a parity discrete topoloy. Assuming a parity discrete

topology <Td> and the operator PARITY, a complete axiom system for join dependencies on <Td> is given as follows:

THEOREM[3.3]. A complete axiom system for join dependencies on a parity discrete topology <Td> is given as follows:

G1: $(X,Y) \in M[Td] \Longrightarrow \{X \cup Y,\ X \cup \overline{Y}\} \in G[Td]$,

G2: $\{P, Q\} \in G[Td]$, $ROOT[P] \cap ROOT[Q] \neq \emptyset$
$\Longrightarrow (ROOT[P] \cap ROOT[Q],\ ROOT[Q]) \in M[Td]$,

G3: $D \in G[Td] \Longrightarrow D \cup \{Q\} \in G[Td]$,

G4: $D \cup \{P,Q\} \in G[Td]$, $P \subset Q \Longrightarrow D \cup \{Q\} \in G[Td]$,

G5: $D \cup \{Q\} \in G[Td]$, $E \in G[Td]$,
$Vk = \{ a \mid a \in EVEN[Ei \cap Ej],\ ODD[Ei \cap Ej] = \emptyset,$
for all pairs $Ei, Ej \in E,\ i \neq j \}$,
$ROOT[Vk] \cap ROOT[ODD[Q]] = \emptyset$,
$V = \{ P \mid P = Ep \cap (Q \cup Vk),\ Ep \in E \}$
$\Longrightarrow D \cup V \in G[Td]$,

where $X, Y \in Td$, $P, Q \in$ <Td>, $D, E \subseteq$ <Td>,

Td: a discrete topology,

<Td>: a parity topology of the discrete topology Td,

G[Td]: a set of join dependencies on <Td>. <>

PROOF: See APPENDIX[A.3]. <>

EXAMPLE[3.3]. Considering EXAMPLE[2.5], we have a set U of attributes: $U = \{a,b,c,d,e\}$, and

an initial set W of join dependencies for $W \subseteq G[Td]$:

$W = \{J1,J2,J3,J4\}$, where

$J1 = \{\{a,b,c\}, \{c,d,e\}\}$, $J2 = \{\{a,d\}, \{a,e\}, \{b,c,e\}\}$,

J3 = {{b,d}, {a,b,c,e}}, J4 = {{a,b,e}, {a,c,d}}.

Then, we can derive join dependencies, J5, J6, J7, and J8 by the following applications of the axiom system. By applying axiom G5 to J2 and J3,

PARITY[J2] = {{a-,d+}, {a-,e+}, {b-,c+,e+}},

Q = {a-,d+},

E = PARITY[J3] = {{b+,d+}, {a-,b+,c+,e+}},

Vk = {b+},

V = {{b+,d+}, {a-,b+}}, and then

J5 = D ∪ V

= {{b+,d+}, {a-,b+}, {a-,e+}, {b-,c+,e+}}.

By applying axiom G5 to J5 and J4,

J6 = D ∪ V

= {{b+,d+}, {a-,b+}, {a-,e+}, {a+,b-,e+}, {a+,c+}}.

By applying axiom G3 and G4 to J6,

J7 = {{b+,d+}, {a-,b+,e+}, {a+,b-,e+}, {a+,c+}}.

By applying axiom G5 to J1 and J7,

PARITY[J1] = {{a+,b+,c+}, {c+,d+,e+}},

Q = {c+,d+,e+},

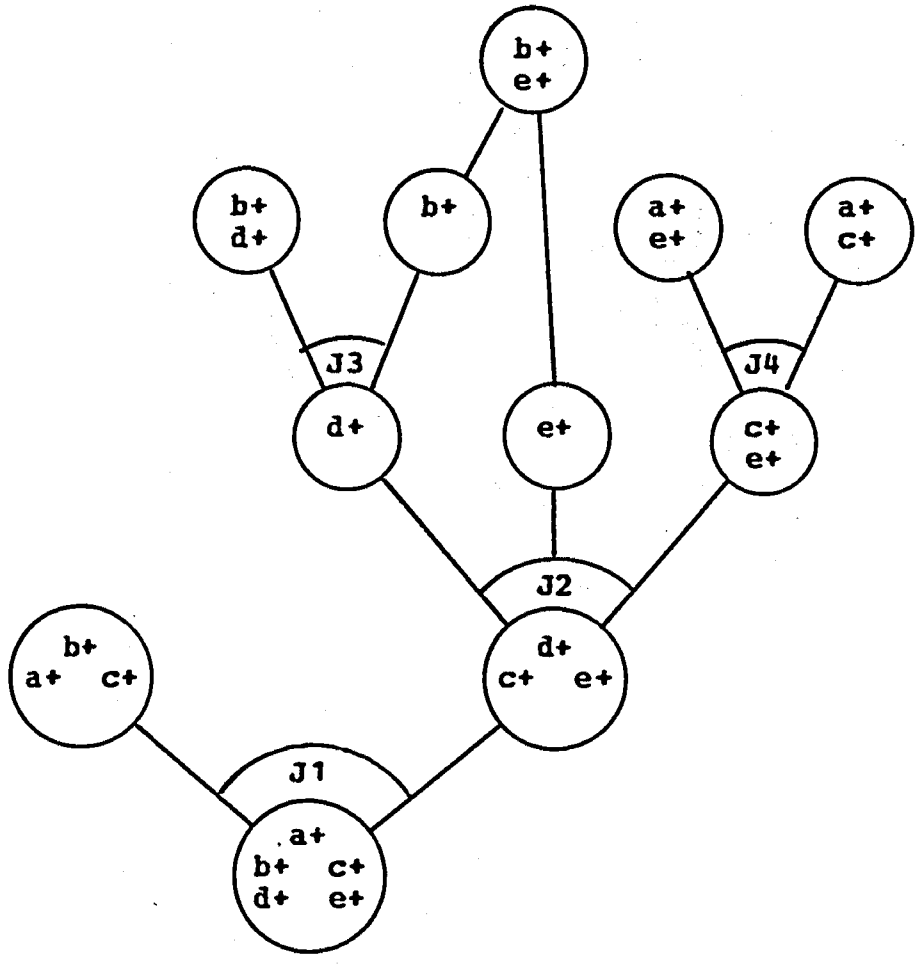
E = J7,

Vk = {a+,b+},

V = {{b+,d+}, {b+,e+}, {a+,e+}, {a+,c+}}, and then

J8 = {{a+,b+,c+}, {b+,d+}, {b+,e+}, {a+,e+}, {a+,c+}}.

Since G[Td] is given on a parity topology, deparitized join dependencies from J5, J6, J7, J8 represent a set of all possible paritized join dependencies by applying the PARITY operator. FIGURE[3.3] gives the derivation process. <>



FIGURE[3.3]. A successful derivation process.

Axioms, G1, G2 state that multivalued dependencies are a special case of join dependencies. G[Td] is a subset of the power set of the parity discrete topology. Also, G[Td]

is a closed collection on the parity discrete topology <Td> introduced by using the complete axiom system for join dependencies. In the case of an arbitrary partition topology the closed collection G[T] for a topology T can be generalized, because Vk ∈ <T> in the axiom G5. We have examined the complete axiom systems for functional, multivalued, and join dependencies and are ready to look at the process of using the dependencies in the construction of the topology for an abstract space.

[3.3] TOPOLOGIZING THE SET OF ATTRIBUTES

In the previous section, we examined the closed collections: F[Td], M[Td], and G[Td] which are given on a discrete topology Td. Note that F[Td] is a set of functional dependencies, M[Td] is a set of multivalued dependencies, and G[Td] is a set of join dependencies. Also note that the closed collections can be given on arbitrary topologies. An ideal topology, called an optimal topology, is minimal in size but fine enough to accommodate the closed collections. We describe a way to introduce an optimal topology in this section. Recall that for a given set F[T] on an arbitrary topology T, a subset Fs[T] of F[T] is called a cover of F[T] if F[T] is implied by Fs[T]. From the concept of minimal cover, MINCOVERs for P[T], M[T], and G[T] are introduced in

such a way that the MINCOVER is a minimal cover, as given in the following definitions.

DEFINITION[3.2]. MINCOVER[F[T]].

MINCOVER[F[T]] is defined as a subset of F[T], satisfying the following conditions:

1. $(X \cup Z, Y \cup Z) \notin$ MINCOVER[F[T]] for $Z \neq \emptyset$,
2. $(Z,Y) \notin$ MINCOVER[F[T]] if $(X,Y) \in F[T]$ for $X \subset Z$,
3. $(X,Y) \notin$ MINCOVER[F[T]] if $(X,Z) \in F[T]$ for $Y \subset Z$,
4. F[T] is implied by MINCOVER[F[T]],

where $X, Y, Z \in T$, T: an arbitrary topology. <>

EXAMPLE[3.4]. Consider a topological space (U,T):

U = {a,b,c,d,e},

T: an arbitrary topology which accommodates the initially given set of functional dependencies, such that

$$(\{a,b\},\{b,c\}),\ (\{c\},\{d\}),\ (\{b,c\},\{d\}) \in F[T].$$

Then, $(\{a,b\},\{c,d\}),\ (\{c\},\{d\}) \in$ MINCOVER[F[T]]. <>

DEFINITION[3.3]. MINCOVER[M[T]].

MINCOVER[M[T]] is defined as a subset of M[T], satisfying the following conditions:

1. $(X \cup Z, Y \cup Z) \notin$ MINCOVER[M[T]] for $Z \neq \emptyset$,
2. $(Z,Y) \notin$ MINCOVER[M[T]] if $(X,Y) \in M[T]$ for $X \subset Z$,
3. $(X, Y \cup Z) \notin$ MINCOVER[M[T]]

   if $(X,Y), (X,Z) \in$ MINCOVER[M[T]],
4. M[T] is implied by MINCOVER[M[T]],

where $X, Y, Z \in \tilde{T}$,

$\tilde{T}$: a partition topology of an arbitrary topology T. <>

EXAMPLE[3.5]. Consider a topological space (U,T):

$U = \{a,b,c,d,e\}$,

T: an arbitrary topology which accommodates the initially given set of multivalued dependencies, such that

$(\{a,b\},\{b,c\}), (\{c\},\{d\}), (\{b,c\},\{d\}) \in M[T]$.

Then, we can get

$(\{a,b\},\{c\}), (\{c\},\{d\}), (\{a,b\},\{d\}) \in$ MINCOVER[M[T]]. <>

Both MINCOVER[P[T]] and MINCOVER[M[T] minimize the left side while MINCOVER[P[T]] maximizes the right side and MINCOVER[M[T]] minimizes the right side from the conditions given in their definitions. In the relational database, assuming the selected topology is a discrete topologyy Td, the concepts of elementary functional dependency and elementary multivalued dependency were introduced in the paper by Zaniolo-Melkanoff[1981]. A functional dependency is called an elementary dependency if it has the form

$(X,\{p\})$ where $p \not\in X$ and $(Y,\{p\}) \not\in P[Td]$ where $Y \subset X$.

A multivalued dependency (X,Y) is called an elementary dependency if Y is disjoint from X and

$(A,B) \not\in M[Td]$ where $A \subset X$ and $B \subset Y$.

Both elementary functional dependencies and elementary multivalued dependencies have minimum left side and minimum right side. Thus, the collection of elementary multivalued

dependencies computed from M+ is MINCOVER[M[Td]]. However, MINCOVER[P[Td]] can be computed by maximizing the right side from the collection of elementary functional dependencies from F+.

DEFINITION[3.4]. MINCOVER[G[T]].

MINCOVER[G[T]] is defined as a subset of G[T], satisfying the following conditions:

1. $A \cup \{X\} \not\in$ MINCOVER[G[T]]
   if $A \cup \{Y,Z\} \in G[T]$ for $X = Y \cup Z$,
2. $A \not\in$ MINCOVER[G[T]] if $B \in$ MINCOVER[G[T]] for $B \subset A$,
3. G[T] is implied by MINCOVER[G[T]],

where $X, Y \in \tilde{T}$, $A \subseteq \tilde{T}$,

$\tilde{T}$: a partition topology of an arbitrary topology T. <>

The first condition minimizes the size of each element of a join dependency. The second condition minimizes the size of a join dependency. Thus, in the sense of minimal size, MINCOVER[G[T]] is a minimal cover of G[T].

EXAMPLE[3.6]. Consider a topological space (U,T):

U = {a,b,c,d,e},

T: an arbitrary topology which accommodates the initially given set of join dependencies, such that

{{a,b,c},{c,d},{c,e}}, {{a,b,c},{c,d,e}},

{{a,b,c},{b,c},{c,d,e}} $\in$ G[T].

Then, {{a,b,c},{c,d},{c,e}} $\in$ MINCOVER[G[T]].

Note that {{a,b,c},{c,d,e}} violates condition 1 and {{a,b,c},{b,c},{c,d,e}} violates condition 2. <>

MINCOVER[F[T]], MINCOVER[M[T]], and MINCOVER[G[T]] are unique minimal covers of F[T], M[T], and G[T], respectively, since they are constructed by minimization and maximization. The uniqueness can be directly recognized by examining the conditions given in their definitions. Only remaining consideration is that the counterpart of an open set must be included for the unique minimal cover of multivalued dependencies. Now, we can define optimality for the selection of the topology as follows:

DEFINITION[3.5]. OPTIMAL TOPOLOGY.

For the given F[T], M[T], and G[T] on an arbitrary topology T, the subbase for an optimal topology To is defined as follows:

(X,Y) ∈ MINCOVER[F[T]] ==> X, Y ∈ SUBBASE[To],

(X,Y) ∈ MINCOVER[M[T]] ==> X, Y ∈ SUBBASE[To],

{ A1, A2, ..., An } ∈ MINCOVER[G[T]]

==> A1, A2, ..., An ∈ SUBBASE[To]. <>

EXAMPLE[3.7]. Considering EXAMPLE[3.4],

({a,b},{c,d}), ({c},{d}) ∈ MINCOVER[F[T]].

Thus, {a,b}, {c,d}, {c}, {d} ∈ SUBBASE[To]. <>

Note that from the definition, an optimal topology is

fine enough to accommodate the closed collections. Thus, the structure of closed collections can be introduced into the optimal topology. Observe that under the assumption, the optimal topology is the finest topology, which is a discrete topology, the topological database will be exactly the same as the relational database. In the next section, we examine a set of structures introduced on the optimal topology.

[3.4] STRUCTURES ON TOPOLOGY

[3.4.1] CONSTRAINT STRUCTURES

After obtaining the optimal topology for the set of attributes, structures called strong, weak, and join constraints, can be introduced on this topological space. The three types of constraints are abstract structures redefined from the functional, multivalued, and join dependencies. We formalize the definition of the constraint structures through the notion of complete axiom systems.

DEFINITION[3.6]. STRONG CONSTRAINT ON OPTIMAL TOPOLOGY.

A strong constraint, denoted S[To] on the optimal topology To is a subset of the product topology of To, satisfying the

complete axiom system for the set of functional dependencies on the optimal topology. <>

EXAMPLE[3.8]. Considering EXAMPLE[3.7], we have

MINCOVER[F[T]] = { ({a,b},{c,d}), ({c},{d}) } and

SUBBASE[To] = { {a,b}, {c,d}, {c}, {d} }.

Then, we get an optimal topology To:

To = { $\emptyset$, U, {a,b,c,d}, {a,b,c}, {a,b,d}, {a,b}, {c,d}, {c}, {d} } and

a strong constraint S[To] on To:

S[To] = { (U,U), (U,{a,b,c,d}), ... (U,{d}), ({a,b,c,d},{a,b,c,d}), ... ({a,b,c,d},{d}), ... ({c,d},{d}), ({a,b},{c,d}), ({a,b},{c}), ({a,b},{d}), ({c},{d}), (U,$\emptyset$), ($\emptyset$,$\emptyset$) }. <>

DEFINITION[3.7]. WEAK CONSTRAINT ON OPTIMAL TOPOLOGY.

A weak constraint, denoted W[To] on the optimal topology To is a subset of the product topology of partition topology of To, satisfying the complete axiom system for the set of multivalued dependencies on the optimal topology. <>

DEFINITION[3.8]. JOIN CONSTRAINT ON OPTIMAL TOPOLOGY.

A join constraint, denoted J[To] on the optimal topology To is a subset of the power set of the partition topology of To, satisfying the complete axiom system for the set of join dependencies on the optimal topology. <>

Constraints, S[To], W[To], and J[To] are the redefinitions on an optimal topology from the set of dependencies in the relational database. The advantage of these redefinitions is the fact that they do not use the entity level rules. In other word, they are defined without using the collection of tuples. The definitions are given directly by the complete axiom systems on the topological space without concerning the instances. Recall that a database space D is completely specified by DIM[D], TOP[D], and INS[D]. Constraint structures are defined on the topological space (DIM[D],TOP[D]), ignoring INS[D] for the database space D. Dependency closures, F[T], M[T], and G[T] are introduced on an arbitrary topology T. Thus, in the case of T = To,

S[To] = F[To], W[To] = M[To], and J[To] = G[To].

From now on, we will use the notations, S[T], W[T], and J[T] for strong, weak, and join constraint. In this case, we assume that the topology T is always optimized unless we specify otherwise.

[3.4.2] COMPUTING THE STRUCTURES

In the previous section, we examined the basic concept of minimal cover for F[T], M[T], and G[T] on an arbitrary topology T. An optimal topology To was computed from the MINCOVER which was shown to be a minimal cover. For any given set of W of functional dependencies, F[Td] is computed

on the discrete topology Td. By introducing an optimal topology To for F[Td], S[To] can be computed on the optimal topology To. From the definition of MINCOVER[F[T]] for an arbitrary topology T, we can get the following result for the set W:

MINCOVER[F[Td]] = MINCOVER[S[To]].

This argument says the fact there are algorithms to compute the MINCOVERs of S[To], W[To], and J[To] on the optimal topology To, when a set of dependencies is given on the discrete topology Td. The following algorithms for the mincovers are created directly from their definitions.

ALGORITHM[3.1]. MINCOVER[S[To]].

INPUT: a set of functional dependencies;

OUTPUT: MINCOVER[S[To]];

PROCESS:

[1] TEMP <--- INPUT;

[2] while there exists an element (X∪Z, Y∪Z) of TEMP
do delete (X∪Z, Y∪Z) from TEMP,
if Y ≠ ∅ insert (X∪Z, Y) into TEMP;

[3] while there exists a pair (X,Y), (Z,Y) of TEMP where X ⊂ Z
do delete (Z,Y) from TEMP;

[4] while there exists a pair (X,Y), (V,Z) of TEMP where V ⊆ Y
do insert (X,Z) into TEMP;

[5] while there exists a pair (X,Y), (X,Z) of TEMP
do delete (X,Y), (X,Z) from TEMP,
insert (X, Y∪Z) into TEMP;
[6] while there exists a pair (X,Y), (X,Z) of TEMP
where Y ⊂ Z
do delete (X,Y) from TEMP;
[7] OUTPUT <--- TEMP;
[8] halt;
END OF MINCOVER[S[To]]. <>

EXAMPLE[3.9]. Considering EXAMPLE[3.4] in Section[3.3], we can examine ALGORITHM[3.1]. Then, the set of functional dependencies, given initially, is

{ ({a,b},{b,c}), ({c},{d}), ({b,c},{d}) }.

These are an input of ALGORITHM[3.1]. Then, in step [2], delete ({a,b},{b,c}) and insert ({a,b},{c}). In step [3], delete ({b,c},{d}). Then, in step [4], insert ({a,b},{d}). In step [5], delete ({a,b},{c}), ({a,b},{d}) and insert ({a,b},{c,d}). In step [6], delete nothing. Finally, we get an output of ALGORITHM[3.1]:

MINCOVER[S[To]] = { ({a,b},{c,d}), ({c},{d}) }. <>

ALGORITHM[3.2]. MINCOVER[W[To]].

INPUT: a set of multivalued dependencies;
OUTPUT: MINCOVER[W[To]];
PROCESS:
[1] TEMP <-- INPUT;

[2] for all elements (X,Y) of TEMP
do insert (X, $\overline{X \cup Y}$) into TEMP;
[3] while there exists an element (X∪Z, Y∪Z)
of TEMP
do delete (X∪Z, Y∪Z) from TEMP,
if Y ≠ ∅ insert (X∪Z, Y) into TEMP;
[4] while there exists a pair (X,Y), (Z,Y) of TEMP
where X ⊂ Z
do delete (Z,Y) from TEMP;
[5] while there exists a pair (X,Y), (X,Z) of TEMP
do delete (X, Y∪Z) from TEMP;
[6] OUTPUT <--- TEMP;
[7] halt;
END OF MINCOVER[W[To]]. <>

ALGORITHM[3.3]. MINCOVER[J[To]].
INPUT: a set of join dependencies;
OUTPUT: MINCOVER[J[To]];
PROCESS:
[1] TEMP <--- INPUT;
[2] while there exists an element Ji of TEMP
such that A⊂B for A, B ∈ Ji
do delete Ji from TEMP,
insert Ji - {A} into TEMP;
[3] while there exist element Ji, Jj of TEMP
such that A⊆B, B ∈ Jj for all A ∈ Ji
do delete Jj from TEMP;

[4] OUTPUT <--- TEMP;

[5] halt;

END OF MINCOVER[J[To]]. <>

THEOREM[3.4]. ALGORITHMs [3.1], [3.2], and [3.3] compute correctly their MINCOVERs. <>

PROOF: Considering ALGORITHM[3.1], each step does not violate the condition being a cover for the inputed set of functional dependencies. The step [3] minimizes left side. The steps [4], [5], and [6] maximize right side. Thus, the result is a minimal cover for the inputed set of functional dependencies. Recall that an optimal topology is given by the MINCOVER[F[T]]. Considering the MINCOVER[F[T]] is a subset of F[T], we recognize the algorithm correctly gives the result MINCOVER[S[To]]. The same argument can be applied to ALGORITHM[3.2] and ALGORITHM[3.3]. <>

Consider an arbitrary set of dependencies which is initially given. ALGORITHM[3.1], ALGORITHM[3.2], and ALGORITHM[3.3] compute MINCOVER[S[To]], MINCOVER[W[To]], and MINCOVER[J[To]], respectively. Then, an optimal topology can be given by the mincovers from the definition. Recall that a subbase of optimal topology can be directly chosen by the mincovers. Thus, a given arbitrary set of dependencies, an optimal topology To for the set of dependencies can be computed through the mincovers as follows:

ALGORITHM[3.4]. SUBBASE[To].

INPUT: MINCOVER[S[To]], MINCOVER[W[To]],
MINCOVER[J[To]];

OUTPUT: SUBBASE[To];

PROCESS:

[1] TEMP <--- MINCOVER[S[To]], MINCOVER[W[To]];

[2] while there exist an element (X,Y) of TEMP
do insert X, Y into SUBBASE[To],
delete (X,Y) from TEMP;

[3] TEMP <--- MINCOVER[J[To]];

[4] while there exist an element
{ A1, A2, ..., An } of TEMP
do insert A1, A2, ..., An into SUBBASE[To],
delete { A1, A2, ..., An } from TEMP;

[5] OUTPUT <--- SUBBASE[To];

[6] halt;

END OF SUBBASE[To]. <>

THEOREM[3.5]. ALGORITHM[3.4] computes correctly a subbase of optimal topology for the given constraints. <>

PROOF: Obvious by DEFINITION[3.5]. <>

Since all of basic features for the database design unit in the topological database model are defined, we close this chapter by giving a comprehensive example of a database selected from geographical data, in order to illustrate the topological database model as follows:

River				Metropolis			
River	Length	Discharg	Outflow	Metropol	State	Area	Populati
Mississi	2350	642000	Gulf	St-Louis	Missouri	2380	2216000
				Memphis	Tennesse	820	843000
				New-Orle	Louisian	810	1175000
				Minneapo	Minnesot	2140	1978000
Missouri	2500	76000	Mississi	Kansas-C	Missouri	1490	1254000
				Omaha	Nebraska	470	548000
Ohio	1300	258000	Mississi	Louisvil	Kentucky	990	881000
				Cincinna	Ohio	1820	1476000
				Pittsbur	Pennsylv	2330	2165000
Arkansas	1450	45000	Mississi	Wichita	Kansas	470	367000
				Tulsa	Oklahoma	1170	569000
				Little-R	Arkansas	610	380000

FIGURE[3.4]. A sample data set II.

EXAMPLE[3.10]. A database space D is defined from the geographical data, as given in FIGURE[3.4], in the following way. DIM[D], the dimension of the database space D, is given by a set of all attributes as follows:

{ R, L, C, O, M, S, A, P }, where

R: a set of rivers,

L: a set of positive integers (length of the river),

C: a set of positive integers (discharge of the river),

O: a set of outflows of the rivers,

M: a set of metropolises located on the bank of the river,

S: a set of states,

A: a set of positive integers (area of the metropolis),

P: a set of positive integers
(population of the metropolis),

This database space is a 8-dimensional space. FIGURE[3.5] shows the dimension of a 8-dimensional space. The semantics of the data constraints gives us functional and multivalued dependencies, such that

$$(\{R\},\{L,C,O\}),\ (\{M,S\},\{A,P\}) \in F[T],$$

$$(\{R\},\{M,S,A,P\}) \in M[T].$$

Then, we have MINCOVER[S[To]] by ALGORITHM[3.1]:

$$\{\ (\{R\},\{L,C,O\}),\ (\{M,S\},\{A,P\})\ \},$$

MINCOVER[M[To]] by ALGORITHM[3.2]:

$$\{\ (\{R\},\{M,S,A,P\})\ \}.$$

Note that we consider only the original members of M[T] in this case. From this fact, an optimal topology To is computed by a subbase of the topology, such that

{R}, {L,C,O}, {M,S}, {A,P} ∈ SUBBASE[To].

TOP[D], a topology of the database space D, is computed by the SUBBASE[To] as follows:

{ ∅, DIM[D], {R}, {L,C,O}, {M,S}, {A,P}, {R,L,C,O},
{R,M,S}, {R,A,P}, {L,C,O,M,S}, {L,C,O,A,P},
{M,S,A,P}, {R,L,C,O,M,S}, {R,L,C,O,A,P}, {R,M,S,A,P},
{L,C,O,M,S,A,P} }.

Note that TOP[D] is an optimal topology To and SUBBASE[To] is a minimal base of TOP[D], given in FIGURE[3.6]. INS[D], a set of 12 points in the database space D, is given in FIGURE[3.7]. Finally, a strong constraint S[To] is given as follows:

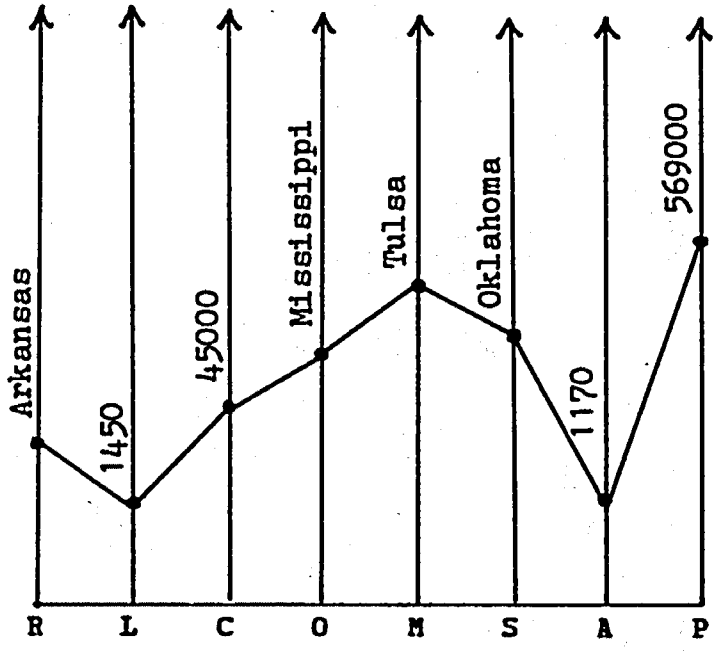
{ ({R},{L,C,O}), ({M,S},{A,P}), ({R,M,S},{L,C,O,A,P}),
({M,S,A,P},{M,S}), ({M,S,A,P},{A,P}),.....
({L,C,O,M,S},{A,P}), ({R,L,C,O},{L,C,O}),.....
(DIM[D],DIM[D]), (DIM[D],{M,S,A,P}),.....
(∅,∅), (DIM[D],∅), ({R},∅),....., ({M,S,A,P},∅) }.

M[To] can be computed in a similar manner. <>

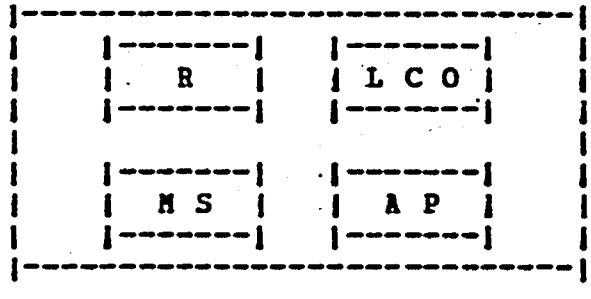
Recall that we assume a discrete topology in the case of relational database model. In the example above, for the discrete topology T, we have F[T] as follows:

F[T] = S[To] ∪ { ({L,C,O},{L,C}), ({L,C,O},{C,O}),
({L,C,O},{L,O}), ({L,C,O},{L}),.....
({L,C},{L}), ({L,C},{L}), ({L,C},{C}),..... }.

Comparing F[T] to S[To], we can observe that F[T] includes lots of trivial dependencies.



FIGURE[3.5]. A 8-dimensional Database Space.



FIGURE[3.6]. A Subbase of a Topology.

```
(Mississippi,2350,642000,Gulf,
                St-Louis,Missouri,2380,2216000)

(Mississippi,2350,642000,Gulf,
                  Memphis,Tennessee,820,843000)

(Mississippi,2350,642000,Gulf,
              New-Orleans,Louisiana,810,1175000)

(Mississippi,2350,642000,Gulf,
             Minneapolis,Minnesota,2140,1978000)

(Missouri,2500,76000,Mississippi,
              Kansas-City,Missouri,1490,1254000)

(Missouri,2500,76000,Mississippi,
                       Omaha,Nebraska,470,548000)

(Ohio,1300,258000,Mississippi,
                  Louisville,Kentucky,990,881000)

(Ohio,1300,258000,Mississippi,
                    Cincinnati,Ohio,1820,1476000)

(Ohio,1300,258000,Mississippi,
            Pittsburgh,Pennsylvania,2330,2165000)

(Arkansas,1450,45000,Mississippi,
                       Wichita,Kansas,470,367000)

(Arkansas,1450,45000,Mississippi,
                     Tulsa,Oklahoma,1170,569000)

(Arkansas,1450,45000,Mississippi,
                 Little-Rock,Arkansas,610,380000)
```

FIGURE[3.7]. An Instance of a Database Space.

CHAPTER IV

DATABASE EQUIVALENCE

Topologizing the set of attributes, the sets used in the relational model have been replaced with the topologies, in the topological model. We will examine the concept of database equivalence in the topological model through the facility of topologies in this chapter. Since the database space was introduced for the mathematical treatment of the topological view of the data, our task is to find appropriate bijections between the database spaces.

The structure of a database space is specified by the corresponding topological space. Observing the wellknown property of topological equivalence, appropriate bijections which we need to find become homeomorphisms. Thus, being a homeomorphism is a necessary condition for the bijection of our task. For database spaces to be topologically equivalent it is a necessary condition that they are structurally equivalent. The optimal topology introduced in Chapter III reflects the structures on the set. Thus, the procedure to find appropriate bijections of our task can be separated into two tasks. The first task is to determine

homeomorphisms. The second task is to select the appropriate bijections among the homeomorphisms. An algorithm for determining homeomorphisms showing the database equivalence is given in ALGORITHM[4.1].

In Section[4.1] we introduce the definition of database equivalence. Definitions for the topological equivalence and constraint equivalence are described. The base graphs for the topologies are used to construct the appropriate homeomorphisms to show the topological equivalence. In Section[4.2], we examine the base graph for a topology to find the appropriate homeomorphisms. At the end of this chapter, we briefly lay the foundation for the database generator. We believe the concept of database equivalence is the critical first step in achieving database design automation.

[4.1] EQUIVALENCE OF DATABASE SPACES

The concept of database space equivalence is formally defined in this section. Recall that whenever two topological spaces are topologically equivalent there exists a set of homeomorphisms. Since the constraint structure is given on the optimal topology, the concept of equivalence of constraint structures can be defined by using the appropriate homeomorphism as follows:

DEFINITION[4.1]. CONSTRAINT EQUIVALENCE OF DATABASE SPACES.

Database spaces, D1, D2 are called strong constraint equivalent for strong constraints, S1[TOP[D1]], S2[TOP[D2]] if and only if there exists a homeomorphism:

$$h: DIM[D1] \longrightarrow DIM[D2]$$

for the topological spaces:

(DIM[D1],TOP[D1]), (DIM[D2],TOP[D2]), such that

if $(A1,B1) \in S1[TOP[D1]]$

then $(h[A1],h[B1]) \in S2[TOP[D2]]$ and

if $(A2,B2) \in S2[TOP[D2]]$

then $(h^{-1}[A2],h^{-1}[B2]) \in S1[TOP[D1]]$. <>

Formally in the same mannner, weak constraint and join constraint equivalence can be defined. Observe that if database spaces, D1, D2 are constraint equivalent then the corresponding two topological spaces, (DIM[D1],TOP[D1]) and (DIM[D2],TOP[D2]) are topologically equivalent since there exists a homeomorphism,

$$h: DIM[D1] \longrightarrow DIM[D2].$$

In this discussion, let us call database spaces, D1, D2 are topologically equivalent if the corresponding topological spaces are topologically equivalent. Thus, for the given database spaces, being topologically equivalent is a necessary condition for being constraint equivalent, but it is not a sufficient condition. This necessary condition gives us an intermediate step for the classification of equivalent database spaces. The concept of topological

equivalence of database spaces plays important role in the algorithm for testing the constraint equivalence of database spaces in the next section.

[4.2] DETERMINING THE HOMEOMORPHISMS

A procedure for finding the appropriate homeomorphisms is based on the base graphs for topologies. Recall that the definition of base graph is given in DEFINITION[2.7]. It is briefly reexamined here. Considering a topological space (X,T), let B be the minimal base and C be the collection of intersections for elements of B. Then, the set {X}∪B∪C can be expressed as a connected acyclic digraph, called the base graph for topology T, in the following way. The vertex set is a set {X}∪B∪C and the edge set is a set of the partial orders of inclusion properties for the members of set {X}∪B∪C excluding the transitive orders. The weight of a vertex is the size of set. The base graph has only one source which is X. Also, note that transitive edges, such as A1->A3 in the case of A1->A2, A2->A3, are eliminated from the edge set.

From elementary graph theory, recall that a bigraph is a graph whose vertex set V can be partitioned into two subsets V1 and V2 such that every edge of the bigraph joins V1

with V2. A bigraph is nontrivial if

$$2 \leq |V1| \quad \text{and} \quad 2 \leq |V2|.$$

If a bigraph contains every edge joining V1 and V2, then the bigraph is a complete bigraph. For example, a graph (V,E), where V = {A1,A2,B1,B2} and

E = {(A1,B1), (A1,B2), (A2,B1), (A2,B2)}, is a complete bigraph. Then, the properties of a base graph for a topology can be summarized in the following remark.

REMARK[4.1]. Let (X,T) be a topological space. Then, the base graph (V,E) for T has the following properties:

(1) It is unique for the topology T.

(2) The size of vertex set is bounded by $2|X|$.

(3) It does not include
a nontrivial complete bigraph (Vs,Es)
as a subgraph such that
$Vs = V1 \cup V2$, $V1 \cap V2 = \emptyset$, and
for $(a,b) \in E$,
$a \in V1$ if and only if $b \in V2$.

(4) It does not include transitive edges. <>

PROOF:

(1) Because a minimal base, intersections, and inclusion properties of sets are unique if the sets are finite.

(2) By THEOREM[2.4].

(3) Assume that a subgraph (Vs,Es) satisfying the given conditions exists. Let

$V1 = \{ a1,a2,....,ai \}$ and

V2 = { b1,b2,.....,bj }.

Then, $b1 \cap b2 \cap \ldots \cap bj = a1 \cup a2 \cup \ldots \cup ai$ and

$b1 \cap b2 \cap \ldots \cap bj$ is a minimax intersection.

Thus, (Vs,Es) does not exist.

(4) By the definition, transitive orders of inclusion are eliminated. <>

Since the base graph for a topology is unique, topological spaces are topologically equivalent if and only if the base graphs for topologies are equivalent. Both the vertex set and the edge set have linear bound by the properties (2), (3), and (4). Considering two topological spaces and two corresponding base graphs, an isomorphism for the two base graphs is a homeomorphism for the two topological spaces, because the vertex set of base graph is a base of the corresponding topology. An algorithm for testing the equivalence of base graphs is not hard to be found from REMARK[4.1], as discussed briefly below.

Two base graphs (V1,E1), (V2,E2) are isomorphic if there exists a bijection h mapping V1 to V2, such that, for any $p,q \in V1$, $(p,q) \in E1$ if and only if $(h[p],h[q]) \in E2$.

An automorphism of a graph is an isomorphism of the graph onto itself. The automorphism partition of a graph consists of cells satisfying the condition that two vertices are in the same cell if and only if there exists an isomorphism from the graph to itself mapping the first vertex to the

second. The graph isomorphism problem is equivalent to the graph automorphism partition problem, as given in Fowler-et al[1983]. To see this, first construct a graph which is the disjoint union of the two given base graphs which are to be tested for isomorphism. Then, construct the graph automorphism partition for the disjoint union. The original two base graphs are isomorphic if and only if there exists one cell of the automorphism partition containing a vertex from each of the original base graphs. This is a brief discussion on the graph automorphism partition. Interested reader can look at the Fowler-et al[1983] paper for details of the general algorithm.

A procedure for the base graph automorphism partition is described briefly below. Vertices in the same cell are said to be indistinguishable while vertices in different cells are said to be distinguishable. Edges (p,q), (u,v) are said to be indistinguishable if both vertices p, u are indistinguishable and vertices q, v are indistinguishable. Otherwise, they are said to be distinguiahable. Differences of vertices are depth, weight, the number of direct edges, and the number of inverse edges. Initially, vertices of the base graph are partitioned into the different cells by their differences. Then, a recursive partition procedure can be applied by the distinguishable edges as follows:

```
procedure AUTOMORPHISM-PARTITION(graph);
  { COMMENT: Input is the disjoint union of the two given
    base graphs which are to be tested for equivalence. }
  procedure GRAPH-PARTITION(cells, distinguishable-edges);
      do  GRAPH-PARTITIONING by distinguishable edges;
      if  graph partitioned
    then  mark distinguishable direct edges
          call  GRAPH-PARTITION(cells, direct-edges)
          mark distinguishable inverse edges
          call  GRAPH-PARTITION(cells, inverse-edges)
    else  null;
  end of GRAPH-PARTITION;
  main procedure
       do  GRAPH-PARTITIONING by vertex differences;
     call  GRAPH-PARTITION(vertex-set-cells, edge-set);
    check  symmetric ring graph;
  end of AUTOMORPHISM-PARTITION.
```

EXAMPLE[4.1]. Let (X,T) be a topological space, where

X = {a,b,c,d,e,f} and

BASE[T] = { {a},{a,b},{a,b,c},{d,e},{d,e,f} }.

Then, a base graph (V,E) for T:

V = { X,{a},{a,b},{a,b,c},{d,e},{d,e,f} }

E = { ({a,b,c},X), ({d,e,f},X), ({a,b},{a,b,c}),
({a},{a,b}), ({d,e},{d,e,f}) }.

Considering only this base graph, AUTOMORPHISM-PARTITION will give the following computations.

Initial cells of vertices:

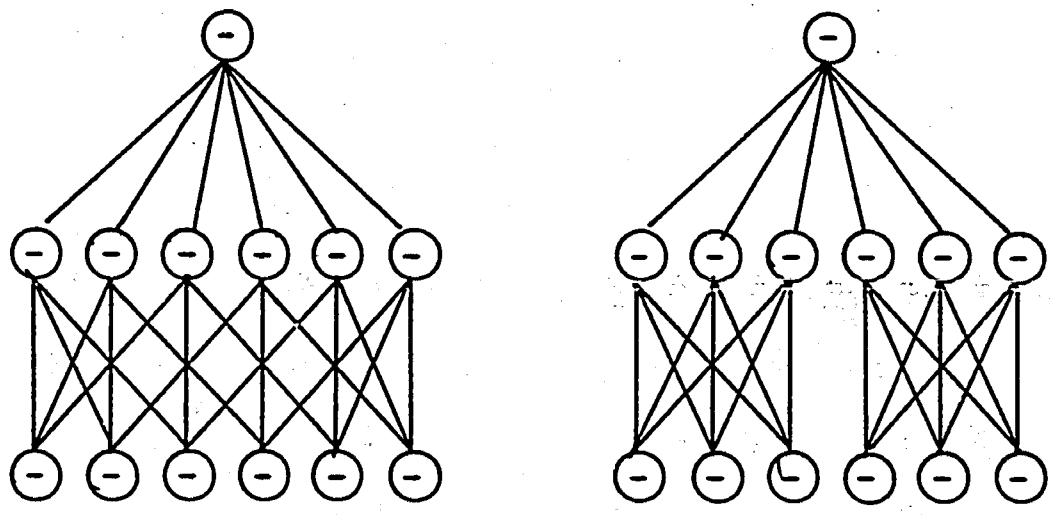
{X}, [{a,b,c},{d,e,f}], [{a,b}], [{d,e}], [{a}].

After partitioning, final cells of vertices:

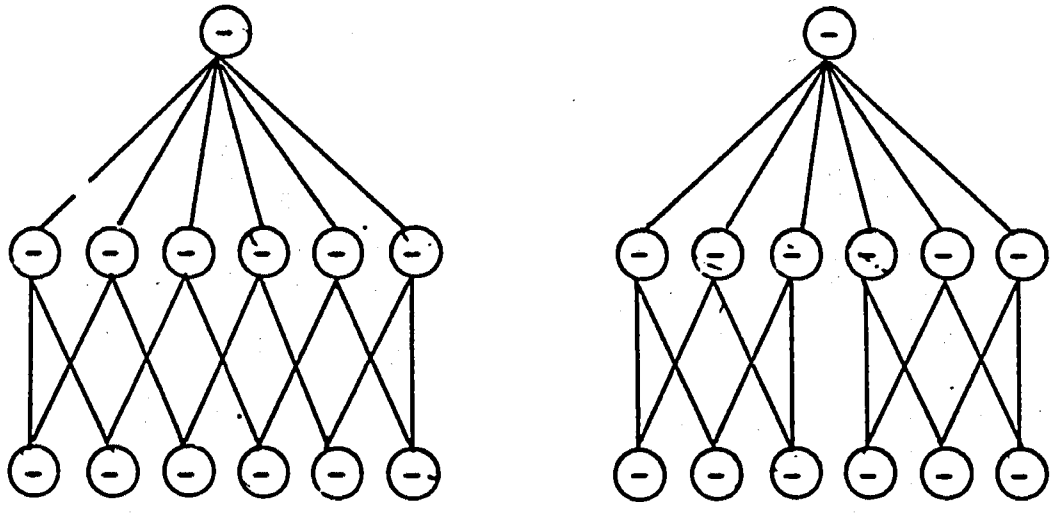
{X}, [{a,b,c}], [{d,e,f}], [{a,b}], [{d,e}], [{a}].

Note that the cell {{a,b,c},{d,e,f}} was partitoned into the two cells [{a,b,c}] and [{d,e,f}] by the distinguishable edges ({a,b},{a,b,c}) and ({d,e},{d,e,f}). <>

Note that if a graph includes a nontrivial complete bigraph as a subgraph as given in FIGURE[4.1], then this partition procedure fails. Note that a base graph can not include a nontrivial complete bigraph. The other interesting problem is the inclusion of a symmetric ring graph as a subgraph of the base graph. An example of including symmetric ring graph is given in FIGURE[4.2]. The partition procedure also fails in this case. This condition can exist for base graphs and after applying the partition procedure, this situation must be checked. Since each partitioning process has n as a bound and the maximum number of recursive calls is n, the complexity of the partition procedure has an n^2 bound. We have described a procedure to determine the homeomorphisms between two topological spaces. A topological structure of a topological space can be identified by its base graph. The automorphism partition procedure gives a partition of a base graph to show indistinguishable vertices. The correctness of this process can be summarized in the following theorem.



FIGURE[4.1]. Including nontrivial complete bigraphs.



FIGURE[4.2]. Including symmetric ring graphs.

THEOREM[4.1]. The base graph automorphism partition procedure correctly determines homeomorphisms. <>

PROOF: Let G1, G2 be base graphs for topologies, T1, T2, respectively to be tested for homeomorphism. By applying the graph automorphism partition procedure to the disjoint union of the base graphs, G1, G2, we have a set of cells. Recall that a cell is a set of indistinguishable vertices which means that two vertices are in the same cell if and only if there exists an isomorphism from the graph to itself mapping the first vertex to the second. For each cell, if we create an automorphism mapping a vertex in G1 to a vertex in G2, then this automorphism is an isomorphism mapping G1 to G2, because two vertices in the same cell are indistinguishable. Since a vertex is an open set, an isomorphism mapping G1 to G2 is a homeomorphism mapping T1 to T2 by the definition of homeomorphism. Thus, the automorphism created is a homeomorphism. <>

We have determined a set of homeomorphisms by the vertex partitioning of base graphs. Those homeomorphisms show only the topological equivalence of database spaces. To show the constraint equivalence of database spaces, we need the appropriate bijections for indistinguishable vertices which are in the same cell. Appropriate bijections can be determined by introducing the structure of scopeset into the partitioned base graph. A scopeset for a strong constraint S[T], denoted by SCOPESET[S[T]], is a set of scopes for S[T]

such that a scope is X∪Y if (X,Y) ∈ MINCOVER[S[T]].

The formal definition of the scopeset is given in the next chapter. Note that a head of scope and a tail is a set of vertices by their definitions. Scopes are indistinguishable if both heads and tails are indistinguishable. Otherwise, they are distinguishable. Initially, a scopeset can be partitioned by putting indistinguishable scopes into the same cell. Since vertices are distinguishable if they belong to distinguishable head or tail, the partitioned scopeset provides a way for further partitioning the partitioned base graph. Thus, we have an algorithm to determine the homeomorphisms showing the strong constraint equivalence of database spaces as follows:

ALGORITHM[4.1]. DATABASE-SPACE-EQUIVALENCE.

INPUT: base graphs, G1, G2 for topologies, T1, T2,.
strong constraints, S1[T1], S2[T2];

OUTPUT: false or
true with homeomorphisms
showing strong constraint equivalence of
database spaces, (X1,T1), (X2,T2);

PROCESS:

[1] create graph by disjoint union of G1, G2;

[2] call AUTOMORPHISM-PARTITION(graph);

[3] create union of SCOPESET[S1[T1]], SCOPESET[S2[T2]];

[4] do SCOPESET-PARTITIONING
by distinguishable scopes;

```
[5]    mark  distinguishable vertices;
[6]      do  GRAPH-PARTITIONING
                          by distinguishable vertices;
[7]      if  exist
       then  OUTPUT  <--  create isomorphism
                          mapping G1 to G2
                          with concerning
                               SCOPESET[S1[T1]],
                               SCOPESET[S2[T2]]
       else  OUTPUT  <--  false;
[8]  halt;
END OF DATABASE-SPACE-EQUIVALENCE.   <>
```

THEOREM[4.2]. ALGORITHM[4.1] determines correctly existing homeomorphisms showing the strong constraint equivalence of the database spaces. <>

PROOF: By applying the graph automorphism partition procedure, we have a set of cells. A cell is a set of indistinguishable vertices in the sense of the base graph structure by THEOREM[4.1]. Applying further partition by the scopeset structures for strong constraints, vertices in the same cell are indistinguishable in the sense of both the base graph structure and the scopeset structure. Thus, the created automorphism, as given in THEOREM[4.1], is a homeomorphism showing the strong constraint equivalence of database spaces. <>

EXAMPLE[4.2]. Consider database spaces, D1, D2, where

DIM[D1] = {a,b,c,d,e,f,g,h,i},

DIM[D2] = {p,q,r,s,t,u,v,w,x}.

and sets of functional dependencies, F1, F2, where

F1 = {({a},{b,c,d}),({b},{c}),({e},{f,g,h,i}),({f},{g})},

F2 = {({p},{q,r,s}),({q},{r}),({t},{u,v,w,x}),({u},{v})}.

Then, we have

SUBBASE[TOP[D1]] = {{a},{b,c,d},{b},{c},{e},
{f,g,h,i},{f},{g}},

SUBBASE[TOP[D2]] = {{p},{q,r,s},{q},{r},{t},
{u,v,w,x},{u},{v}}.

The disjoint union of base graphs for the automorphism partition is given in FIGURE[4.3].

After graph partition, we have 5 cells:

{{a},{e},{p},{t}}, {{b,c,d},{q,r,s}},

{{b},{c},{q},{r}}, {{f,g,h,i},{u,v,w,x}},

{{f},{g},{u},{v}}.

After scopeset partition, we have 3 cells:

{{a:b,c,d},{p:q,r,s}},

{{b:c},{f:g},{q:r},{u:v}},

{{e:f,g,h,i},{t:u,v,w,x}}.

Note that notation {head:tail} is used for a scope.

By repartitioning the cells of graph, we have 8 cells:

{{a},{p}}, {{e},{t}}, {{b},{q}}, {{c},{r}},

{{f},{u}}, {{g},{v}}, {{b,c,d},{q,r,s}},

{{f,g,h,i},{u,v,w,x}}.

Finally, with concerning the strong constraint, we have

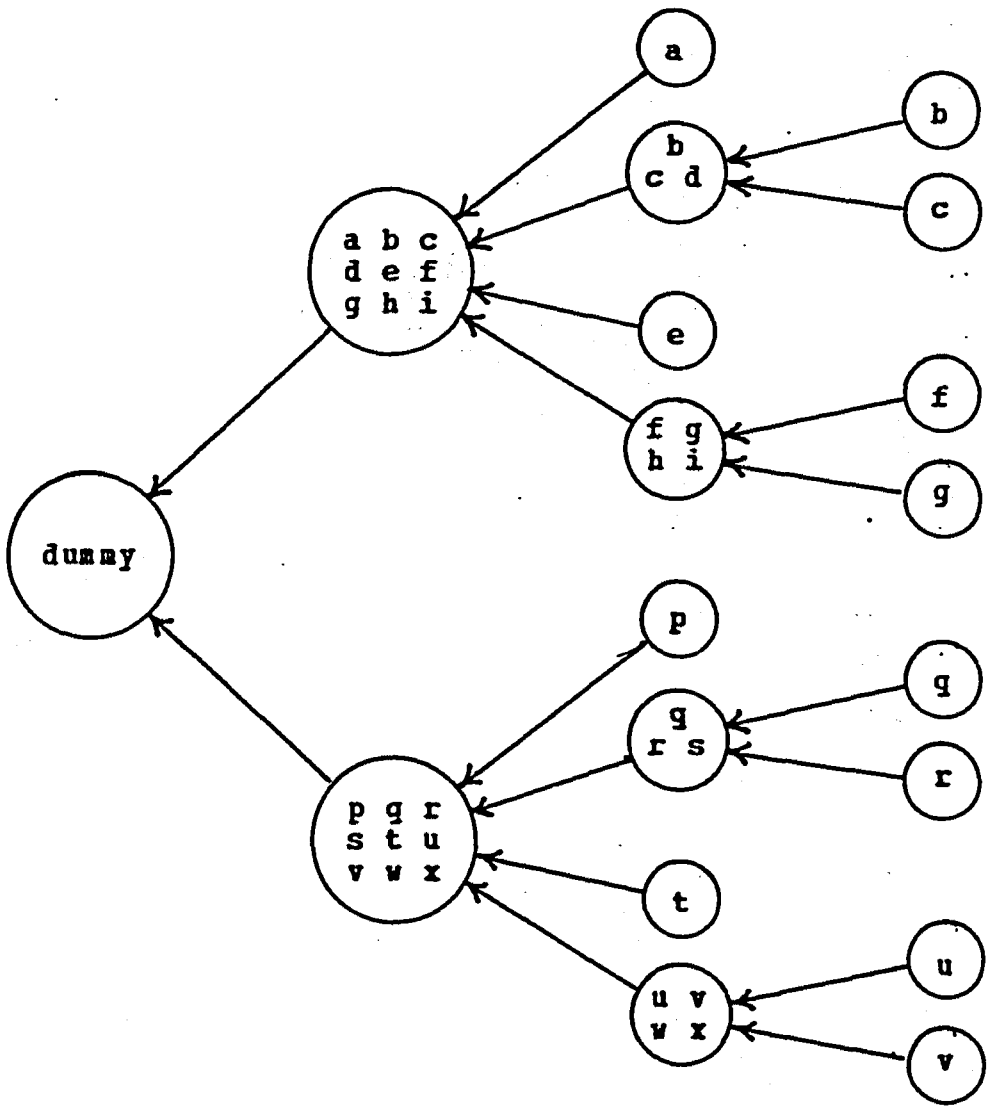
isomorphisms showing the strong constraint equivalence:

{a} --> {p}, {b} --> {q}, {c} --> {r},

{d} --> {s}, {e} --> {t}, {f} --> {u},

{g} --> {v}, {h,i} --> {w,x}.

Our task is done. <>



FIGURE[4.3]. An example of a graph for partition.

Note that ALGORITHM[4.1] shows only the strong constraint equivalence. However, this algorithm conceptually can be extended to include the weak constraint and the join constraint.

[4.3] REMARKS ON A DATABASE GENERATOR

The ultimate goal of database design automation is to build a database generator which is an automatic database design machine. A database generator is an algorithm such that input is available information, like data and structures, and output is a database design which preserves the available information.

Databases are classified into the equivalent classes for the construction of database generator by using the concept of database equivalence. Thus, the function of database generator can be represented as a 4-dimensional space, called a database generator space. The coordinates of the database generator space represent the database class which is classified into the equivalent class. The 4 coordinates are ATTRIBUTES, TOPOLOGIES, CONSTRAINTS, and DECOMPOSITIONS. ATTRIBUTES-coordinate is a set of attributes, TOPOLOGIES-coordinate is a set of topologies, CONSTRAINTS-coordinate is a set of constraints, and DECOMPOSITIONS-coordinate is a set

of decompositions.

Databases are classified, first by the number of attributes, second by the introduced topologies, third by the given constraints, finally by the applied decomposition processes. In general, the theory of a database generator is to study the properties of database generator space. While the examination of the details of the database generator are beyond the scope of this discussion, the concept is important to pointing out our interest database equivalence.

CHAPTER V

DATABASE DECOMPOSITION

A central idea of the database design is the notion of decomposition in the relational database model. In general, complex information is to be expressed as a superposition of simpler components without missing information. We describe the notion of decomposition for the topological database model in this chapter.

Section[4.1] includes the concept of subspace for the decomposition processes. Section[4.2] introduces manipulation operators to handle the subspaces. In the decomposition process, two plausable conditions are known as lossless join and dependency preservation properties. Section[4.3] describes the concepts of scopes and neighborhoods for constraint preserving decompositions. Finally, scopeset structures help the tie breaking task for the symmetric vertices of the base graph when we are trying to show constraint equivalence.

[5.1] SUBSPACE

Recall the concept of subspace, given in Section[2.1]. For the readers convenience the basic notation is reexamined here. For a database space D, if t $\in$ INS[D] and X $\subseteq$ DIM[D], then t[X] is used to represent the X-component of the tuple t and PROJECT[INS[D]:X] is used to represent the collection of the X-component of the tuples in INS[D]. The notation PROJECT[TOP[D]:X] is used for a relative topology of TOP[D] on X, as given in Section[2.1]. Thus, the notation PROJECT is defined as follows:

PROJECT[TOP[D]:X] = { Xi | Xi = Yi$\cap$X, Yi $\in$ TOP[D] }.

By using this notation, the concept of a subspace of a database space is introduced as follows:

DEFINITION[5.1]. SUBSPACE OF DATABASE SPACE.

Let D be a database space. Then, a database space S is a subspace of D if and only if

DIM[S] $\subseteq$ DIM[D] and

TOP[S] = PROJECT[TOP[D]:DIM[S]]. <>

EXAMPLE[5.1]. Consider a database space D:

DIM[D] = {a,b,c,d},

TOP[D] = {$\emptyset$, DIM[D], {c}, {d}, {c,d}, {a,b},
{a,b,c}, {a,b,d}}.

Then, considering a database space S:

DIM[S] = {a,b,c},

TOP[S] = {Ø, DIM[S], {c}, {a,b}},

we can observe S is a subspace of D. <>

Since if n = |X| then a database space (X,T) is a n-dimensional space, a subspace always has a dimension smaller than or equal to the dimension of original space. Recall an instance of a n-dimensional database space is a set of n-tuples. If an instance of a database space D, INS[D] is carried into a subspace S of it, then an instance of the database space S, INS[S] is given as

INS[S] = PROJECT[INS[D]:DIM[S]].

EXAMPLE[5.2]. Considering EXAMPLE[3.1], an example of subspace S is

DIM[S] = {CITY,STATE}, TOP[S] = {Ø,DIM[S]}.

For this subspace S, a 2-dimensional figure and INS[S] are given in FIGURE[5.1]. <>

Subspaces play the major role in database design since they are the basic database unit. Considering a topology T on the universal set U, space (U,T) is called a microcosmic database space. The formal definition for the microcosmic database space will be given in the next section. The major problem of database design is to find the way to represent a microcosmic database space by using a set of subspaces of the microcosmic space. Recall that a database space D is completely specified by three things: DIM[D], TOP[D], and

INS[D]. In the assumption of a microcosmic database space, specification of DIM and TOP is necessary since join constraints on the microcosmic database space must be provided to the user for the design level. However, specification of INS[D] is not necessary since data constraints are defined by the notion of the attribute set without using the notion of the record level instance.

Recall the concepts of intra-record type and inter-record type, as given in Section[2.1], for database representation. For the readers convenience, an example of those types from EXAMPLE[3.10], is given in FIGURE[5.2]. Now, we want to take those concepts into the database space representation. In the database space D, intra-record type can be represented by DIM[D] and inter-record type can be represented by TOP[D]. Since the optimal topology is constructed from data constraints such as dependencies, the inter-record types, derived from data constraints, must be included in TOP[D]. The usefulness of TOP as inter-record type check can be shown in the following example.

EXAMPLE[5.3]. Consider DIM[D] = {a,b,c,d,e,f} and functional dependencies, ({a},{b,c,d}), ({d},{e,f}). Then, a topology TOP[D] is

TOP[D] = {$\emptyset$, X, {a}, {d}, {b,c,d}, {e,f}, {a,d}, {a,e,f}, {a,b,c,d}, {d,e,f}, {b,c,d,e,f}, {a,d,e,f}}.

Now, we can observe that {a,b,c,d} and {b,e,f} are a poor

decomposition in the sense they do not have lossless join property since $\{a,b,c,d\} \cap \{b,e,f\} = \{b\} \notin TOP[D]$.

In the notion of relational database, we can observe that this implies that $(\{b\}, \{a,b,c,d\})$ and $(\{b\}, \{b,e,f\})$ are not functional dependencies. <>

Note that TOP gives a necessary condition to check for a poor decomposition, as given in the example above. We will examine this feature in more detail in Section[5.3]. The other interesting feature is the comparison between the optimal and the discrete topology in the practical database environment. In the practical database design, increasing the size of intra-record type is increasing the size of open set in the optimal topology. This structure is quite common in a practical database environment, witnessed by employee information. For example, we have functional dependencies, such as

(employee, employee name),

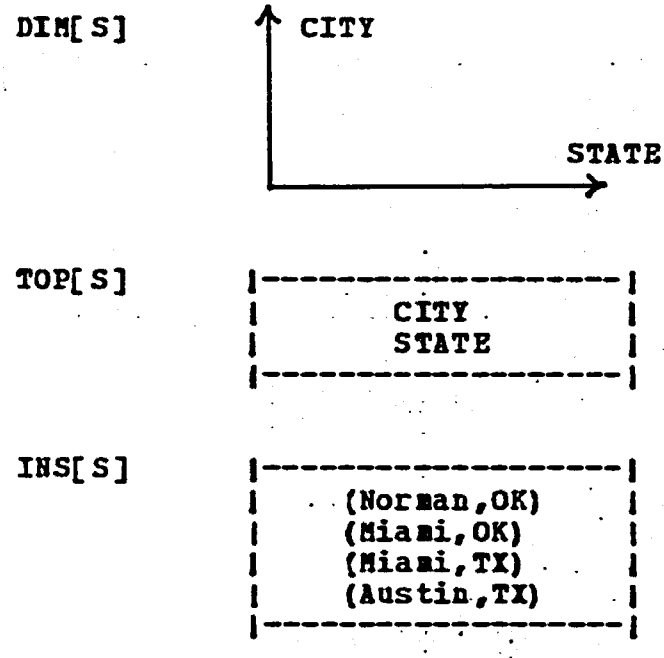
(employee, address),

(employee, educational background),

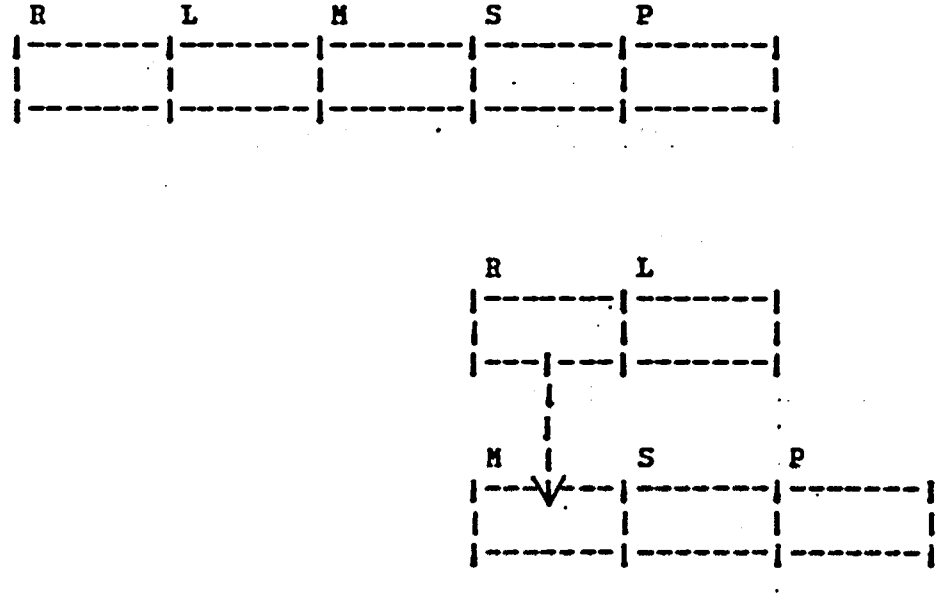
(employee, department),

(employee, salary).

In this case, a set {employee name, address, educational background, department, salary} can be an intra-record type, making it an open set in the optimal topology. This reduces the number of open sets required in the optimal topology for many practical applications.



FIGURE[5.1]. An example of a Subspace.



FIGURE[5.2]. Intra-record and inter-record types.

[5.2] MANIPULATION OPERATORS

In the previous section, by introducing the concept of subspace, we illustrated that the topological database can be represented by a set of database spaces. To manipulate the set of database spaces, manipulation operators must be defined. Since database spaces have new facilities such as TOP which are not available in the relational database, new operators are needed. The data operator DOP and the structure operator SOP are defined as basic manipulation operators. Note that our intention is not in the syntax of operators but in the semantics of operators at this moment.

DEFINITION[5.2]. DATA OPERATOR (DOP).

Syntax:

DOP[ACCESS/ CREATE/ FILTER]

ACCESS: { D1, D2, D3, ..., Dl }

CREATE: { a1, a2, a3, ..., am }

FILTER: { C1, C2, C3, ..., Cn }

Semantics:

ACCESS: a set of database spaces,

CREATE: a set of attributes,

FILTER: a set of conditions,

DOP[ACCESS/ CREATE/ FILTER]: database space, where

DIM[DOP] = CREATE,

SUBBASE[TOP[DOP]] = { X | X = PROJECT[TOP[Di]:Si],

Si = CREATE$\cap$DIM[Di], Di $\in$ ACCESS },

INS[DOP] = { t | t[Si] ∈ PROJECT[INS[Di]:Si],
Si = CREATE∩DIM[Di], Di ∈ ACCESS,
t satisfies FILTER }. <>

DEFINITION[5.3]. STRUCTURE OPERATOR (SOP).

Syntax:

SOP[ACCESS/ OBJECT/ SQUERY]

ACCESS: { D1, D2, D3, ..., Dl }

OBJECT: { a1, a2, a3, ..., am }

SQUERY: { Q1, Q2, Q3, ..., Qn }

Semantics:

ACCESS: a set of database spaces,

OBJECT: a set of attributes,

SQUERY: a set of queries,

SOP[ACCESS/ OBJECT/ SQUERY]: a series of trues and falses selected from element of SQUERY as follows:

True: if OBJECT has the property indicated by SQUERY,

False: if OBJECT does not have. <>

Considering a manipulation operation Wi where ACCESS[Wi] = { D1, D2, D3, ..., Dn }, the domain of the operation Wi, denoted DOM[Wi], is defined as

DOM[Wi] = DIM[D1]∪DIM[D2]∪.....∪DIM[Dn]. Then,

CREATE[Wi] or OBJECT[Wi] ⊆ DOM[Wi]. /

Considering a set W of operations where

W = { W1, W2, W3, ..., Wm }, the domain of the set W, denoted DOM[W], is defined as

$DOM[W] = DOM[W1] \cup DOM[W2] \cup \ldots\ldots \cup DOM[Wm]$.

For a collection W of operations, we can introduce the concept of microcosmic database space as follows:

DEFINITION[5.4]. MICROCOSMIC DATABASE SPACE (MDBSP).

Considering a database space U and a set W of manipulation operations, the database space U is a microcosmic database space for the set W of operations if and only if

$$DOM[W] \subseteq DIM[U].$$ <>

Operator DOP has three operands: ACCESS, CREATE, and FILTER. ACCESS is given as a set of database spaces, CREATE as a set of attributes, and FILTER as a set of conditions. General description on the process of the DOP operation is expressed in the following way. First DOP will access all relevant database spaces from the operand ACCESS, then create new database space D, such that a dimension of D, DIM[D] is defined by operand CREATE. A topology of D, TOP[D] is computed by a subbase of TOP[D] which is the union of all topologies from the subspaces $DIM[D] \cap DIM[Di]$ for all Di in operand ACCESS. An instance of D, INS[D] is a collection of all tuples satisfying the conditions in operand FILTER, such that $DIM[D] \cap DIM[Di]$ component of the tuple in INS[D] is equal to the same component of certain tuple in INS[Di] for instances of all database spaces in operand ACCESS. Conditions in operand FILTER can be expressed by using comparison and boolean operators such as

>, <. =, union, intersection, and complement.

EXAMPLE[5.4]. As illustration of the DOP operator, the following examples can apply to the database space D, given in EXAMPLE[3.10]:

E = DOP[{D}/ {R,L,C,O}/ $\not\emptyset$];

F = DOP[{D}/ {R,M,S,P}/ { 2000000<P }];

G = DOP[{E,F}/ {R,L,M}/ { 2000<L }].

For the readers convenience, database spaces, E, F, and G are given as follows:

DIM[E] = {R,L,C,O},

TOP[E] = {$\not\emptyset$, DIM[E], {R}, {L,C,O}},

INS[E] = { (Mississippi,2350,642000,Gulf),
(Missouri,2500,76000,Mississippi),
(Ohio,1300,258000,Mississippi),
(Arkansas,1450,45000,Mississippi) },

DIM[F] = {R,M,S,P},

TOP[F] = {$\not\emptyset$, DIM[F], {R}, {M,S}, {P}, {R,M,S}, {R,P},
{M,S,P} },

INS[F] = { (Mississippi,St-Louis,Missouri,2216000),
(Ohio,Pittsburgh,Pennsylvania,2165000) },

DIM[G] = {R,L,M},

TOP[G] = {$\not\emptyset$, DIM[G], {R}, {L}, {M}, {R,L}, {L,M}, {R,M} },

INS[G] = { (Mississippi,2350,St-Louis) }. <>

In order to examine whether DOP can manipulate fully database spaces, operator DOP can be compared with the rela-

tional algebra as given below. INS[DOP] is

cartesian product if $Di \cap Dj = \emptyset$,

natural join if $Di \cap Dj \neq \emptyset$, and

intersection if $Di = Dj$ for Di, Dj ACCESS.

Thus, INS[DOP] operates in the same power as the power of relational algebra. However, in order to manipulate the facility of topologies, an operation TOP[DOP] is introduced. This is the main reason for not simply adopting relational algebra.

The result of an operator SOP is a boolean variable. Examples of SQUERY are: 'Is OBJECT candidate key ?', 'Is OBJECT lossless join ?'. The operator SOP carries out the result whether operand OBJECT selected from the operand ACCESS satisfies the condition specified by operand SQUERY.

[5.3] DEPENDENCY PRESERVATION

The concept of a dependency preserving decomposition is one of the major topics in database theory. This section introduces the matter into the topological database. Let us begin with the precise definition of a constraint preserving decomposition.

DEFINITION[5.5]. S[T] PRESERVING DECOMPOSITION.

Let S[TOP[D]] be a strong constraint on a database space D, S[TOP[P]] on P, and S[TOP[Q]] on Q. Then, P and Q is called a S[TOP[D]] preserving decomposition of D

if and only if DIM[D] = DIM[P]∪DIM[Q]

and there does not exist a set A such that

A ∈ TOP[D], A ∉ TOP[P], B ∈ TOP[Q], B ⊆ C,

and (A,C) ∈ TOP[D]. <>

EXAMPLE[5.5]. Consider a database space D:

DIM[D] = {a,b,c,d},

TOP[D] = { ∅, X, {a,b}, {c}, {a,b,c} }

and an element of S[TOP[D]]: ({a,b},{c}) ∈ S[TOP[D]].

Then, decomposition DIM[P] = {a,c} and DIM[Q] = {b,d} is not a strong constraint preserving decomposition, because in this case

TOP[P] = {∅, DIM[P], {a}, {c}}, {c} ∈ TOP[P],

but {a,b} ∉ TOP[P]. <>

For a more mathematical treatment on the existancy of dependency preserving decomposition, the concept of neighborhood on the database space is introduced along the same fashion as in the topological space. A neighborhood is defined through the concept of scope for the constraint, introduced as follows:

DEFINITION[5.6]. SCOPESET[S[T]].

A set SCOPESET[S[T]] is a set of scopes for S[T] such that

SCOPESET[S[T]] = { S | S = X∪Y,

(X,Y) ∈ MINCOVER[S[T]] }.

In this case, X is called the head of S and Y is called the tail of S. <>

Recall if (X,Y) ∈ S[T] then the scope for (X,Y) is X∪Y in relational database theory. Then, a SCOPESET[S[T]] is a set of all scopes which are derived from the minimal cover of S[T]. Since SCOPESET[S[T]] is a subset of topology T, scopes can be treated as another structure on the topology.

DEFINITION[5.7]. NEIGHBORHOOD.

Considering a topological space (X,T),

if G ∈ SCOPESET[S[T]] and p ∈ G,

then G is called a scope neighborhood of p. A subset N of X is called a neighborhood of p if N is a superset of all scope neighborhoods of p. The collection of all neighborhoods of p ∈ X, denoted Np is called the neighborhood system of p. <>

Note that each attribute has one neighborhood system. Considering two points p, q ∈ X, p and q are separable by the functional dependency preserving decomposition if there exists N ∈ Np such that p ∈ N and q ∉ N. This means that two attributes p and q can be separated by a certain

functional dependency preserving decomposition. We close this chapter by reviewing the general decomposition theory. A topological database is represented by a set of database spaces. A database space can be represented by a set of subspaces through the notion of decomposition. The structures of SCOPESET and NEIGHBORHOOD provide appropriate information for the decomposition process. Recall that topologies are the closed collection under set operations. Decomposition theory can be generalized by applying the set operations on the chosen topologies and by decomposing the database space into the appropriate subspaces.

CHAPTER VI

CONCLUSION

[6.1] CONCLUDING REMARKS

A topological database approach is presented. The database unit of this model is a database space. The database space represents the topological view of the data. A series of structures on the database space is also presented. The defined structures in this discussion are optimal topology, strong constraint, weak constraint, join constraint, scopeset, and neighborhood.

The task to topologize the set of attributes creates the topological structure to the relational database by the result of data semantics analysis. Thus, the topological approach, based on topologies, is expected as more general approach than the relational approach, based on sets. In general, topology provides higher level mathematical sophistication than that of set theory. Under the assumption of the discrete topology, the topological database will be exactly the same as the relational database. One significant benefit of topological structure is to provide an algorithm to test the equivalence of database spaces. We

believe that the concept of database equivalence provides the critical first step towards the database generator.

[6.2] FUTURE RESEARCH

We can anticipate four areas for the future research on the topological database approach. These areas are briefly described in the following sections.

[6.2.1] TOPOLOGICAL REPRESENTATION OF INTER-RECORD TYPES

The properly selected topology provides the facility to store a portion of the semantics of the data. Topologies are able to represent inter-record types in the topological database. The proper representation of record based information can be accomplished by the combination of intra-record type and inter-record type in the database design level. For a database space (DIM[D],TOP[D]), the facilities, DIM[D], TOP[D] are able to represent the combination of the two types. An intra-record type can be reflected by an open set. When we select the collection of intra-record types as a subset of the topology, the topological database model will be a model for design level. We believe future studies must pay much concern to this ability of the topo-

logical database model.

[6.2.2] CLASSIFICATION OF DATABASE SPACES

Many properties of a topological space depend on the distribution of the open sets in the space. If there are few open sets, then the space is more likely to be separable and an arbitrary function on the space to some other space is more likely to be continuous. This fact suggests that there is a certain methodology for the classification of database spaces. The classes of spaces for infinite sets have been well developed in topology literature. Now, we are facing the classes of spaces for finite sets. We can suggest some useful tools from topology literature as follows. A topological pair (X,A) is defined as a topological space X and a subspace A of X. A relative homeomorphism is defined as a map

$$f: (X,A) \longrightarrow (Y,B)$$

of a topological pair (X,A) into a topological pair (Y,B) such that f maps $X \cap \overline{A}$ homeomorphically onto $Y \cap \overline{B}$. This is a relaxation of the continuity condition. A retraction of a space X onto a subspace A of X is a map

$$r: X \longrightarrow A$$

such that $r[a] = a$ for $a \in X$. A subspace A of a space X is a retract in this case. The project operation in a database can be accomplished by a retract. The classifica-

tion process can be provided by establishing the separation axiom system. We believe the separation axiom system for finite spaces can be constructed through the concepts of topological properties such as relative homeomorphism and retraction.

[6.2.3] METRIZATION OF DATABASE SPACE

In a topological space (X,T), ordered pairs of elements in X is called a metric on X if and only if it satisfies the triangular properties for every element in X. If we define certain orders on domains and dimensions of a dadabase space, then we are able to introduce a certain metric to express the relationship between points on the space. A dependency can be replaced with a set of functions. We believe the set of functions, which is given by a dependency, provides proper information for the metrization process.

[6.2.4] DOMAIN ORGANIZATION

In a database space, a coordinate associates a domain which is a finite set of values. A domain can be organized in proper way. If there is many-to-one relationship mapping a domain ai to a domain aj, then the domain ai can be clas-

sified by the domain aj. Furthermore, by maps

f1: {ai} --> {aj} and f2: {ai} --> {ak},

the domain ai is classified in two levels by aj and ak. In general, by maps

g1: {ai,aj} --> {am} and g2: {ai,ak} --> {an},

we have the result that {ai} = {ai,aj} ∩ {ai,ak} has two levels classification. The domain classification provides fast access for certain group of queries. Topology provides the classification levels for domains.

BIBLIOGRAPHY

(Listed in Chronological Order)

[1948] Bourbaki, N., General Topology, Part 1, 2, Elements of Mathematics, Hermann, Paris, France, 1948, English Ed., Addison-Wesley, Reading, MA, 1966.

[1965] Hu, S., Theory of Retracts, Wayne State University Press, Detroit, MI, 1965.

[1966] Dugundji, J., Topology, Allyn and Bacon, Boston, MA, 1966.

[1968] Cullen, H., Introduction to General Topology, D. C. Heath and Company, Boston, MA, 1968.

[1969] Harary, F., Graph Theory, Addison-Wesley, Reading, MA, 1969.

[1970] Codd, E., A Relational Model of Data for Large Shared Data Banks, CACM 1970, Vol.13, No.6, pp. 377-387.

[1971] Data Base Task Group of Conference on Data Systems Languages (CODASYL DBTG), Report, ACM, New York, NY, 1971.

[1974] Armstrong, W., Dependency Structures of Data Base Relationships, IFIP 1974, North-Holland, Amsterdam, Netherland, 1974, pp. 580-583.

[1974] Rustin, R. (editor), Data Models: Data Structure Set versus Relational, Proceedings of 1974 ACM SIGMOD Workshop on Data Description, Access and Control, Vol II, 1974.

[1976] Chen, P., The Entity-Relationship Model: Toward a Unified View of Data, ACM-TODS 1976, Vol.1, No.1, pp. 9-36.

[1976] Michaels, A., Mittman, B., and Carlson, C., A Comparison of the Relational and CODASYL Approaches to Data Base Management, ACM Computing Surveys 1976, Vol.8, No.1, pp. 125-151.

[1977] Beeri, C., Fagin, R., and Howard, J., A Complete Axiomatization for Functional and Multivalued Dependencies, ACM-SIGMOD 1977, pp. 47-61.

[1978] Olle, T., The Codasyl Approach to Database Management, John Wiley and Sons, New York, NY, 1978.

[1978] Steen, L. and Seebach Jr., J., Counterexamples in Topology, 2nd Ed., Springer-Verlag, New York, NY, 1978.

[1979] Codd, E., Extending the Data Base Relational Model to Capture More Meaning, ACM-TODS 1979, Vol.4, No.4, pp. 397-434.

[1979] Kent, W., Limitations of Record-based Information Models, ACM-TODS 1979, Vol.4, No.1, pp. 107-131.

[1979] Maier, D., Mendelzon, A., and Sagiv, Y., Testing Inplications of Data Dependencies, ACM-TODS 1979, Vol.4, No.4, pp. 456-469.

[1981] Beeri, C., Mendelzon, A., Sagiv, Y., and Ullman, J., Equivalence of Relational Database Schemes, SIAM J on Computing 1981, Vol.10, No.2, pp. 352-370.

[1981] Date, C., An Introduction to Database Systems, 3rd Ed., Addison-Wesley, Reading, MA, 1981.

[1981] Zaniolo, C., and Melkanoff, M., On the Design of Relational Database, ACM-TODS 1981, Vol.6, No.1, pp. 1-47.

[1982] Hanson, O., Design of Computer Data Files, Computer Science Press, Potomac, MD, 1982.

[1982] Sciore, E., A Complete Axiomatization of Full Join Dependencies, JACM 1982, Vol.29, No.2, pp. 373-393.

[1982] Ullman, J., Principles of Database Systems, 2nd Ed., Computer Science Press, Potomac, MD, 1982.

[1982] Tsichritzis, D. and Lochovsky, F., Data Models, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[1983] Baik, K. and Miller, L., Topological Approach for Testing Equivalence of Databases, Tech. Report, School of EECS, University of Oklahoma, 1983.

[1983] Fowler, G., Haralick, R., Gray, G., Feustel, C., and Grinstead, C., Efficient Graph Automorphism by Vertex Partitioning, Artificial Intelligence 1983, Vol.21, No.1, North-Holland, Amsterdam, Netherland, 1983, pp. 245-269.

[1983] Maier, D., The Theory of Relational Databases, Computer Science Press, Potomac, MD, 1983.

APPENDICES

APPENDIX[A.1] PROOF OF THEOREM[3.1].

APPENDIX[A.2] PROOF OF THEOREM[3.2].

APPENDIX[A.3] PROOF OF THEOREM[3.3].

APPENDIX[A.1] PROOF OF THEOREM[3.1]

An original work of the proof is in Armstrong[1974]. The proof is accomplished by the following 4 steps.

(I) Define notations as follows:

Fs[Td]: a set of functional dependencies for the discrete topology Td,

F+[Td]: a set of functional dependencies which are logically implied by Fs[Td],

F*[Td]: a set of functional dependencies which are derived from the axiom system by Fs[Td].

Then, for the set of axioms {F1,F2,F3},
we want to prove F*[Td] = F+[Td].

(II) Prove F*[Td] $\subseteq$ F+[Td]: Soundness.

Considering arbitrary X, Y, Z $\in$ Td and t1, t2 $\in$ R,

t1[X] = t2[X] ==> t1[Y] = t2[Y] for Y $\subseteq$ X,

t1[X$\cup$Z] = t2[X$\cup$Z] ==> t1[X] = t2[X], t1[Z] = t2[Z]
==> t1[Y] = t2[Y], t1[Z] = t2[Z]
(because (X,Y) $\in$ F[Td])
==> t1[Y$\cup$Z] = t2[Y$\cup$Z],

t1[X] = t2[X] ==> t1[Y] = t2[Y]
(because (X,Y) $\in$ F[Td])
==> t1[Z] = t2[Z]
(because (Y,Z) $\in$ F[Td])

Thus, by the definition of functional dependency, we get

F*[Td] $\subseteq$ F+[Td].

(III) Prove $F^+[Td] \subseteq F^*[Td]$: Completeness.

For an arbitrary $X \in Td$, define T^*, Dx, as follows:

$T^* = \{ Y \mid (X,Y) \in F^*[Td] \}$,

$Dx = \{ y \mid y \in Y, Y \in T^* \}$.

Consider a set of records $\{ t1, t2 \}$, such that

$t1[Dx] = t2[Dx]$ and $t1[\overline{Dx}] \neq t2[\overline{Dx}]$.

Then, $\{ t1, t2 \}$ satisfies $Fs[Td]$, because

for an arbitrary $(Y,Z) \in Fs[Td]$,

if $Y \subseteq Dx$ then $Z \subseteq Dx$ by axiom F3.

Considering an arbitrary $(X,Z) \notin F^*[Td]$,

$(X,Z) \notin F^*[Td] \Longrightarrow Z \notin T^*$

$\Longrightarrow Z \cap \overline{Dx} \neq \emptyset$

$\Longrightarrow t1[Z] \neq t2[Z]$

$\Longrightarrow (X,Z) \neq F^+[Td]$.

Thus, we get $F^+[Td] \subseteq F^*[Td]$.

(IV) Finally, by the results of (II) and (III), we get

$F^*[Td] = F^+[Td]$. <>

APPENDIX[A.2] PROOF OF THEOREM[3.2]

Beeri-Fagin-Howard[1977] gives an original work of the proof. We will consider only a set of multivalued dependencies. Then, the complete axiom system is involved by axioms, M3, M4, and M5. The proof is accomplished by the following 4 steps.

(I) Define notations as follows:

Ms[Td]: a set of multivalued dependencies
for the discrete topology Td,

M+[Td]: a set of multivalued dependencies which are
logically implied by Ms[Td],

M*[Td]: a set of multivalued dependencies which are
derived from the axiom system by Ms[Td].

Then, for the set of axioms {M1,M2,M3,M4,M5},
we want to prove $M^*[Td] = M^+[Td]$.

(II) Prove $M^*[Td] \subseteq M^+[Td]$: Soundness.

Considering arbitrary X, Y, Z $\in$ Td and
t1, t2, t3, t4 $\in$ R,

$t1[X] = t2[X] \Rightarrow t1[Y] = t2[Y]$
(because $(X,Y) \in P[Td]$)
$\Rightarrow t2[X \cup Y] = t1[X \cup Y]$ and $t2[\overline{X \cup Y}] = t2[\overline{X \cup Y}]$,

$t1[X] = t2[X] \Rightarrow t3[X \cup Y] = t1[X \cup Y]$ and
$t3[\overline{X \cup Y}] = t2[\overline{X \cup Y}]$
(because $(X,Y) \in M[Td]$)
$\Rightarrow t3[\overline{X \cup \overline{X \cup Y}}] = t1[\overline{X \cup \overline{X \cup Y}}]$
(because $\overline{X \cup \overline{X \cup Y}} = \overline{X} \cap (X \cup Y) \subseteq X \cup Y$) and
$t3[X \cup \overline{X \cup Y}] = t2[X \cup \overline{X \cup Y}]$
(because $t3[X] = t1[X] = t2[X]$),

$t1[X \cup Z] = t2[X \cup Z] \Rightarrow t3[X \cup Y] = t1[X \cup Y]$ and
$t3[\overline{X \cup Y}] = t2[\overline{X \cup Y}]$
(because $t1[X] = t2[X]$, $(X,Y) \in M[Td]$)
$\Rightarrow t3[X \cup Y \cup Z] = t1[X \cup Y \cup Z]$

(because $t1[Z] = t2[Z] = t3[Z]$) and

$t3[\overline{X \cup Y \cup Z}] = t2[\overline{X \cup Y \cup Z}]$

(because $\overline{X \cup Y \cup Z} \subseteq \overline{X \cup Y}$),

$t1[X] = t2[X]$ ==>

$t3[X \cup Y] = t1[X \cup Y]$ and $t3[\overline{X \cup Y}] = t2[\overline{X \cup Y}]$

(because $(X,Y) \in M[Td]$) and

$t4[Y \cup Z] = t3[Y \cup Z]$ and $t4[\overline{Y \cup Z}] = t1[\overline{Y \cup Z}]$

(because $t3[Y] = t1[Y]$, $(Y,Z) \in M[Td]$)

==> $t4[\overline{X \cup Y} \cap Z] = t3[\overline{X \cup Y} \cap Z] = t2[\overline{X \cup Y} \cap Z]$

(because $\overline{X \cup Y} \cap Z \subseteq Y \cup Z$, $\overline{X \cup Y} \cap Z \subseteq \overline{X \cup Y}$)

and $t4[Y \cup \overline{Z}] = t1[Y \cup \overline{Z}]$

(because $t4[Y] = t1[Y]$)

==> $t4[X \cup (\overline{Y} \cap Z)] = t2[X \cup (\overline{Y} \cap Z)]$

(because $t4[X] = t2[X]$) and

$t4[\overline{X \cup (\overline{Y} \cap Z)}] = t1[\overline{X \cup (\overline{Y} \cap Z)}]$

(because $\overline{X \cup (\overline{Y} \cap Z)} = \overline{X} \cap (Y \cup Z) \subseteq Y \cup \overline{Z}$).

Thus, by the definition of multivalued dependency we get

$$M^*[Td] \subseteq M+[T].$$

(III) Prove $M+[Td] \subseteq M^*[Td]$: Completeness.

For an arbitrary $X \in Td$, define T^*, Bx, as follows:

$T^* = \{ W \mid (X,Y) \in M^*[Td],\ W = \overline{X} \cap Y \}$,

$Bx = \{ Zi, Zj, Zk \mid P, Q \in T^*,\ Zi, Zj, Zk \neq \emptyset,$

$Zi = P \cap \overline{Q},\ Zj = \overline{P} \cap Q,\ Zk = P \cap Q \}$.

Then, for arbitrary $Zi, Zj \in Bx$,

$Zi \cap Zj = \emptyset,\ Zi \cap X = \emptyset,\ Zj \cap X = \emptyset$.

Consider a set of records { t1, t2, ..., tn } and

$Z1, Z2, \ldots, Zm \in Bx,$

where $m = |Bx|$ and $n = 2^m$, such that

$t1[X] = t2[X] = \ldots\ldots\ldots = tn[X],$

$t1[Z1] = t2[Z1] = \ldots\ldots\ldots = tn/2[Z1],$

$tn/2+1[Z1] = tn/2+2[Z1] = \ldots\ldots\ldots = tn[Z1],$

$t1[Z2] = t2[Z2] = \ldots\ldots\ldots = tn/4[Z2] = tn/2+1[Z2]$

$= tn/2+2[Z2] = \ldots\ldots\ldots = t3n/4[Z2],$

$tn/4+1[Z2] = tn/4+2[Z2] = \ldots\ldots = tn/2[Z2] = t3n/4+1[Z2]$

$= t3n/4+2[Z2] = \ldots\ldots\ldots = tn[Z2],$

$\ldots\ldots\ldots\ldots\ldots$

$\ldots\ldots\ldots\ldots\ldots$

$t1[Zm] = t3[Zm] = t5[Zm] = \ldots\ldots\ldots = tn-1[Zm],$

$t2[Zm] = t4[Zm] = t6[Zm] = \ldots\ldots\ldots = tn[Zm].$

Then, $\{t1,t2,\ldots,tn\}$ satisfies $Ms[Td]$, because

for an arbitrary $(Y,Z) \in Ms[Td]$,

if $\emptyset \neq Z \cap Zi \neq Zi$ for $Zi \in T^*$, then $Y \cap Zi \neq \emptyset$.

Considering an arbitrary $(X,Z) \notin M^*[Td]$,

$(X,Z) \notin M^*[Td] \Rightarrow Z \notin T^*$

$\Rightarrow Z \neq \{ z \mid z \in Zi, \ i \in k \}$

$\Rightarrow (X,Z) \notin M+[Td],$

where k is an arbitrary subset of set $\{1,2,3,\ldots,m\}$.

Thus, we get $M+[Td] \subseteq M^*[Td]$.

(IV) Finally, by the results of (II) and (III), we get

$M^*[Td] = M+[Td]$. <>

APPENDIX[A.3] PROOF OF THEOREM[3.3]

An original work of the proof is from Sciore[1982]. We will consider only a set of join dependencies. Then, the complete axiom system is involved by axioms, G3, G4, and G5. The proof is accomplished by the following 4 steps.

(I) The inference problem has been solved by the chase process. Recall that a successful chase process for join dependencies can be expressed as an acyclic digraph with a sink.

(II) A successful derivation process by the axiom system can be expressed as an acyclic digraph with a sink, as decribed below. In the successful derivation process, each vertex associates to one or more elements of join dependencies. For an edge set G, such that

G = { (Di,X) | Di ∈ D, D: a vertex set, X: a vertex },

the vertex set D associates to a join dependency and the vertex X associates to a substituted vertex. Initial vertex set is given by a join dependency to be tested. The substitution process is given by the axioms, G3, G4, and G5 in such a way that substututes

D to D∪{Q},

D∪{P,Q} to D∪{Q}, and

D∪{Q} to D∪V,

repectively. The sink is a join dependency which is a singleton set.

(III) A successful derivation process can simulate a successful chase process in the same order. Only considerable situation is the case that certain attribute appears more than once in a join dependency but it does not appear in the substituted vertex. Axiom G5 states that this attribute must be paritized with an even parity if the associated component of associated vertex in the successful chase process has even variabble before processing the substitution. Initially, all attributes were paritized with odd parities. Thus, axiom G5 can simulate this situation. In the other situations, a derivation process simply substitutes a vertex set by using a join dependency which is used in the chase process.

(IV) Finally, by the result of (III), for a given join dependency to be tested, there is a successful chase process if and only if there is a successful derivation process by the axiom system. Thus, the axiom system is a complete axiom system for join dependencies. <>