Dasananda, Surapol

THE MULTI-LAYERED SCHEMA OF AN EXTENDED ENTITY-RELATIONSHIP MODEL

The University of Oklahoma PH.D. 1984

THE UNIVERSITY OF OKLAHOMA

GRADUATE COLLAGE

THE MULTI-LAYERED SCHEMA OF AN EXTENDED ENTITY-RELATIONSHIP MODEL

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

BY

SURAPOL DASANANDA

Norman, Oklahoma

1984

THE MULTI-LAYERED SCHEMA OF AN EXTENDED

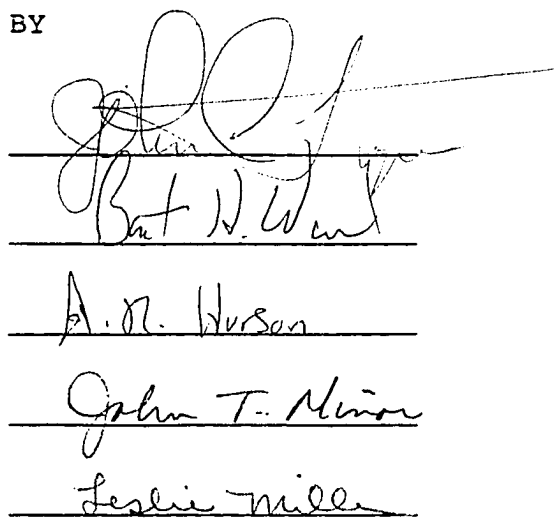
ENTITY-RELATIONSHIP MODEL

A DISSERTATION

APPROVED FOR THE SCHOOL OF ELECTRICAL ENGINEERING

AND COMPUTER SCIENCE

BY



DISSERTATION COMMITTEE

ACKNOWLEDGEMENT

Thank you.

ABSTRACT

In a database design process, the database model used is essential for producing a good conceptual schema. In some database models, like IMS and CODASYL, a conceptual schema contains too many implementation details which complicate the designer's task. The conceptual schema of a relational database hides too much information from the user, because it lacks the neccessary structure. The standard E-R model has more structure and is easy to use. But, it still lacks the ability to express certain types of abstract concepts needed in most design processes. In this work, the E-R Model is extended to include generalization abstraction among different sets of a database. The generalization structures and their components form the conceptual schema of a database. The schema is represented by a multi-leveled diagram. A user is provided with a set of diagram operators to view parts of a diagram relevant to him. A top down design technique based on data dependencies and inapplicability of attributes is also presented for this database model.

CONTENTS

ACKNOWLEDGEMENT iii

ABSTRACT iv

1. Introduction 1
1.1 Objective 1
1.2 Historical Perspective 4

2. The E-R Model 7
2.1 Introduction 7
2.2 Basic Definitions 8
2.3 E-R diagram 11
2.4 Constraints in the E-R Model..... 13

3. Abstraction in Database Model 19
3.1 Introduction 19
3.2 Aggregation and Generalization 20
3.3 Abstraction in the E-R Model 26

4. An Extended E-R Model 33
4.1 Generalization Structures 33
4.2 Attributes and Generalization Structure 50
4.3 Structural Constraints of the Extended E-R Model 56
4.4 Schema and Operators of the Extended E-R Model 66
4.5 Extended E-R Diagram Presentation Technique 76

5. Database Design Technique 86
5.1 Introduction 86
5.2 Designing Process 87
5.3 Categorization Process 91
5.4 Splitting Constraint Consideration 101
5.5 Normalization Process 103
5.6 E-graph and R-graph Analysis 119
5.7 Summary 127

6. Conclusion, Recommendation and Future Works 130
6.1 Conclusion 130
6.2 Recommended Implementation 132
6.3 Future Works 135

REFERENCES 137

APPENDICES 141

CHAPTER ONE

INTRODUCTION

1.1 Objective.

A good design is one of the most important factors in the success of a database system. The result of a database design is a conceptual schema which represents the logical structure of the users' view of a real world system. If the conceptual schema does not faithfully express the semantic information of the real wrold system, then it is of little or no use at all. Thus, at a database design stage, the designers should be provided with a tool which is simple to use, yet powerful enough to capture different types of semantic information in the real world system.

In early 1960, the need to organize related information in several files into an integrated database was recognized. The two prominent database models around that time were the network model (CODASYL) and the hierarchical model (IMS). These two models were created to satisfy the implementors of the early database systems. The data structures provided by these models are linked data structures which tie different types of records and allow them to be accessed together. The conceptual schema of a CODASYL database consists of various plex structures; the conceptual schema of an IMS database consists of different tree structures. In both cases, the information on access paths and data placement is visible to the users in the conceptual schemas. Be-

cause the conceptual schema of an IMS database or a CODASYL database consists of both the logical structure and the physical structure of a database, the design process of these databases is extremely complicated. Novices or casual users must take time to learn the details of a conceptual schema (become experts) before they can use this type of databases.

In 1970, E. F. Codd introduced the relational model based on the set theory in mathematics field. For the first time, the logical structure and the physical structure of a database are separated into the conceptual schema and the internal schema. This separation simplifies the design process tremendously. The mathematical foundation of the relational model has advanced all area of database research. But the recent advance in database theory has also widened the gap between the researchers and the practitioners in the database field. The design theories of the relational model are especially complex and controvesial. In addition, the conceptual schema of a relational database consists solely of relation schemes and has no explicit linkage among them. The lack of structure leads to ambiguities in the conceptual schema. It also increases the chance for the users of a relational database to form meaningless operations.

In 1975, the Entity-Relationship model (E-R model) was introduced by Chen, primarily to provide a communication tool among the user communities and the system analysts of a database system. The logical structure of a real world system is expressed in term of entities (objects) and relation-

ships among them the same way we express our thoughts in simple English sentences. Chen used a diagram called an E-R diagrm to represent the sets of entities and relationship in a database. Several researchers have used this model as a tool for database design process. The diagram produced at the end of a design process can be easily converted into a conceptual schema of other models. The E-R diagram could also be used as the conceptual schema of a database. In this case, we may view the diagram as a relational schema with additional requirement that each relation must represent either a set of entities or a set of explicit linkages (relationships). Even though the E-R model provides a better design tool than the relational model, it still lack the capability to express certain abstract concepts which are relevant in most modeling processes.

In this work, the basic E-R model is enhanced by adding three structures to the basic data types based on generalization abstraction. This makes the extended E-R model more suitable than the standard E-R model as a design tool. By maintaining the basic constructs of the standard E-R model, a database is kept simple and precise. At the same time, additional structures allow more sematic information to be included in a well defined manner. The three abstract structures and the constraints associated with each structure are formally defined using set and graph theories.

Associated with the model is the extended E-R diagram used for representing the logical structure of a database.

This diagram may be used as a design tool or as an actual schema of a database. For a large database, the diagram may be complex and difficult to understand. To simplify a complex diagram, a presentation technique is designed based on the generalization of the diagram itself.

The normalization technique using context dependent dependencies is discussed. The technique is used to modify the logical structure of a database to obtain a "better formed" database. The normalized database will be better reflecting semantic information of the system being modelled and should be more stable over time. Finally, a brief discussion of a possible implementation technique is presented.

1.2 Historical Perspective.

In a database design process, the logical properties of a real world system are captured in the structure of the database schema, the set of constraints and in the sequence of operations performed on the database [D1, W1]. Recently, several researchers have concentrated their effort on adding more data structure to the database schema for better semantic information representation. By making this information visible in the schema, the users may consult the database schema and form database operation accordingly. Most of the data models which include additional data structure in the schema are usually called semantic data models. There have been numerous such models proposed, some of them are presented here.

Abstraction concepts including generalization and aggregation are used to add hierarchical data structures to the schema in the relational model by Smith [S4, S5]. The same principle is also used in the normalization process of the relational model to obtain (3,3) normal form in [S6]. Later on, Smith's semantic model was extended to include procedural attachment and deffered update by Brodie [B8] and Cammarata [C1]. Borkin [B7] did extensive work on the semantic graph model and the semantic relational model. The instance of a database is interpreted by the predicate, case grammar pairs which are included as part of every database schema. He also used these models in database equivalence study. Codd [C8] also proposed an extended relational model called RM/T. In this model, the relation schemas are tied together by different graph structures each of which represent a different abstraction concept. Codd also defined additional relational operators, special null values based on three-valued logic and the system controlled identifiers called surrogates. A new data model called TAXIS was proposed by Mylopolous [B6, M2]. This model is based on abstraction concepts taken from semantic network in AI research. A database in this model consists of tokens, classes and metaclasses which are related by two abstract structures called INSTANCE-OF and IS-A abstractions. The different components of a database are also related by 2 property connections (attributes) called factual property and definitional property. They also defined procedural at-

tachment as special properties of some class or metaclass to indicate the operations allowed as part of the schema itself. A similar approach was introduced by McLeod and King [H2, M1] in their semantic data model called SDM/R. This data model also includes abstract objects such as class, subclass, grouping and interclass connections. Attributes and their inversions were defined on different objects in this model.

The data model used as the basis of the work in this paper is the Entity-Relationship model [C3, C4, C6]. The detailed description of this model is given in the next chapter. Several researchers, e.g., Batini [B1], Scheuermann [S3], Sakai [S1, S2], Dos Santos [D2] and Kent [K2], have proposed extensions to the entity-relationship model by adding some abstraction concept to create more data structure in a database schema. These previous works provide a framework for the model presented in this paper.

In chapter 2, the definition of the original E-R model is examined in detail. In chapter 3, the concepts of abstraction including generalization, aggregation and classification are presented. Several articles on abstraction in E-R model are cited at the end of the chapter. In chapter 4, the structure of the extended model is defined. The structural constraints and the query operators are described. And finally, the normalization technique and the implementation technique are discussed in chapter 5 and chapter 6 respectively.

CHAPTER TWO

THE E-R MODEL

2.1 Introduction.

The Entity-Relationship Model (E-R model) was invented in 1975 by Chen [C3]. The model was intended to be used primarily for the database design process. The part of the real world as viewed by the user community is called the enterprise schema in this model. The enterprise schema created by the model is independent of the DBMS and should be stable over a long period of time. The primary objectives for creating the enterprise schema [C4] are: to serve as documentation for the logical properties of a database, and to serve as a communication tool for designers and various users. The E-R model has gained acceptance from several researchers in database field. Research has been done on different aspects of the model. The evident can be seen in numerous articles appeared in several technical journals since the presentation of the model in 1975 [C2, C5, H1, P1]. Over the past few years E-R model has been considered as a viable model for implementation. There are many databases being implemented using this model but most of these systems are still in the experimental stage [B3, G2, L1].

The E-R model at the conceptual level is a generalized network model without any physical storage or access as-

pects. The model can be easily converted into a network model or a relational model. The E-R model uses a diagram, which consists of nodes and connections, to depict the logical structure of a database. Since most of the semantic information is explicitly stated, the E-R model does not have the semantic ambiguity that the relational model has been criticized for. Because the connections are logical the E-R model can achieve a high level of data independence. The problem of data independence has been a major drawback of the network and the hierarchical models since they have been in existence.

The E-R model consists of two type of nodes, the entity type and the relationship type. The definitions of the two types and the related subjects are as follow.

2.2 Basic Definitions.

2.2.1 Entity and Entity set.

We use e to denote an entity which exists in our mind. An entity may be a physical object that exists in the real world, e.g.,

A T & T,
Jane Doe
Ronald Reagan, and
Mt. Rushmore.

Or an entity may be a concept that may or may not exist physically in the real world, e.g.

transaction number 12335,
a virtue of man, and
King Lear.

What we consider as an entity is highly subjective. And the consideration is based on a designer's view of the world. Usually the world as seen by the designer consists of many entities. These entities are classified into different entity sets denoted by Ei. All the entities that belong to the same entity set have some property in common. Example of entity sets are

PRESIDENT,
PERSON,
EMPLOYEE,
COMPANY, and
SHAKESPEAR CHARACTER.

In the E-R model, Chen also assume the existence of a primitive predicate associated with each entity set to test whether an entity belong to the set or not. For example,

is-a-PRESIDENT(Ronald Reagan) = true,
is-a-COMPANY(Mt. Rushmore) = false, and
is-a-SHAKESPEAR CHARACTER(King Lear) = true.

Note that entity sets may not be mutually disjoint. The entity Jane Doe may belong to entity set PERSON and entity set EMPLOYEE at the same time.

2.2.2 Relationship, Role and Relationship Set.

A relationship set R represents a type of association that exists among a group of entity sets, E1, E2, ..., En. These entity sets may not be distinct. For j=1,2,...,n, Ej plays the role bj in the the relationship R. A member of the relationship set R is an n-tuple which represents a relationship of the type specified by R. The jth component of a tuple is an entity drawn from the entity set Ej. And this

entity also plays the roles bj as dictated by the structure of R. We may write an relationship set R as a set of mappings

```
R = { (b1/e1, b2/e2, ..., bn/en) | e1 E1,...,en En &
                    b1 is the role of E1 in R & ...
                    bn is the role of En in R      }.
```

Each mapping maps roles b1,b2,...,bn into entities e1,e2,...,en which are drawn from E1,E2,...,En respectively. If all roles are explicitly stated, we could also write R as

```
R(b1/E1, b2/E2, ..., bn/En)
            = { (e1,e2,...,en) | e1 E1,..., en En}.
```

Example 2.1

Let PERSON and COMPANY be entity sets.

```
PERSON  = { p1, p2, p3, p4 }
COMPANY = { c1, c2, c3 }
```

Then we may create the relationship sets EMPLOYMENT and MARRIAGE based on these entity sets. The relationship set EMPLOYEE relates PERSON to COMPANY. The relationship set MARRIGE relates PERSON to itself.

```
EMPLOYMENT(EMPLOYEE/PERSON,EMPLOYER/COMPANY) =
                { (p1,c1),(p2,c1),(p3,c2),(p4,c1) }

MARRIAGE(HUSBAND/PERSON,WIFE/PERSON) =
                { (p1,p2),(p3,p4) }.
```

2.2.3 Attribute, Value and Value Set.

A value v is used in the E-R model to describe a characteristic of an entity or a relationship. A value is obtained by measurement or by observation. Like entities and relationships, values are classified into different

value sets Vi based on similarity of value properties, e.g., having the same measurement unit, or describing the same phenomenon, etc. We associate a value to an entity or a relationship through an attribute. An attribute ai is a function from an entity set or a relationship set to a value set or a Cartesian product of value sets.

$$ai : Ei \text{ or } Ri \text{ ----> } V \text{ or } Vi_1 X\ Vi_2 X \ \ldots \ X\ Vi_k.$$

Example 2.2
of attributes and value sets.

Let ID-NUMBER, FIRST-NAME, LAST-NAME and POSITION be the following value sets,

```
ID-NUMBER  = { m | m is an unsigned 3 digit number },
FIRST-NAME = { Bob, Carol, Ted, Alice },
LAST-NAME  = { Smith, Jones, Adams },
POSITION   = { manager, clerk, secretary }.
```

We could then tie these value sets to an entity set or a relationship set as follows:

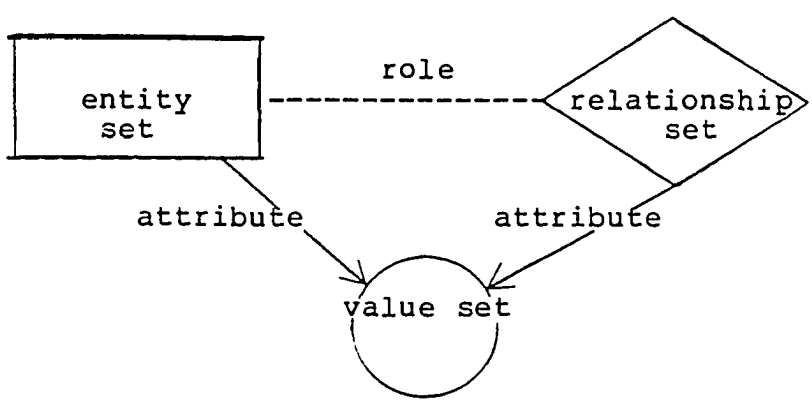
```
ID-NUMBER : PERSON ----> ID-NUMBER,
ID-NUMBER = { (p1,101),(p2,236),(p3,225),(p4,203) },

NAME : PERSON ----> FIRST-NAME X LAST-NAME,
NAME = { (p1,(Bob,Smith)),(p2,Carol,Smith),
           (p3,(Bob,Jones)),(p4,(Alice,Jones)) },

POSITION : EMPLOYMENT ----> POSITION,
POSITION = { ((p1,c1),manager),((p2,c1),clerk),
               ((p3,c2),secretary) }.
```

2.3 The E-R Diagram.

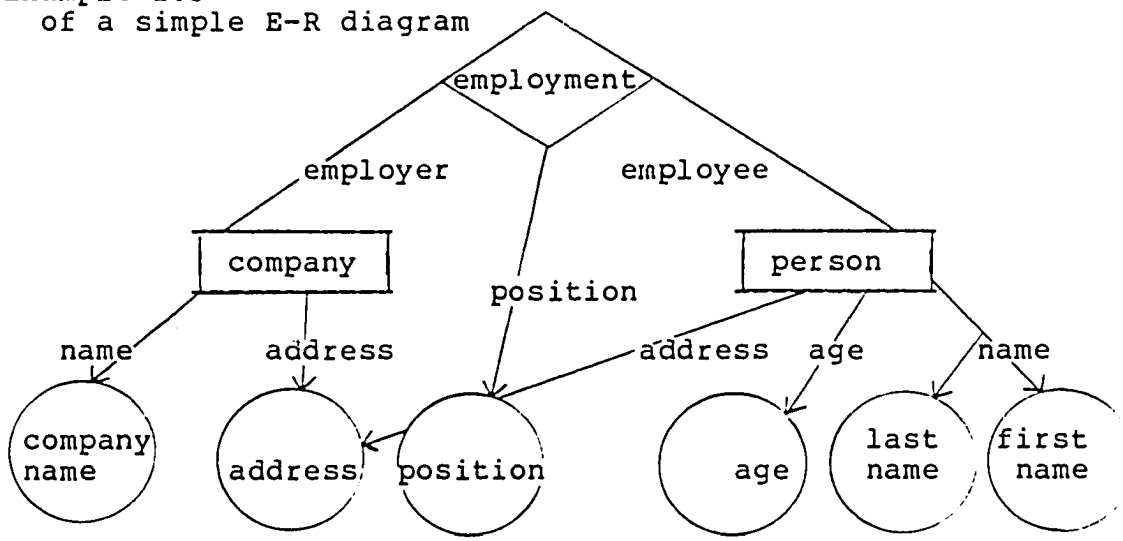
As mentioned earlier, Chen used a diagram to depict the logical structure of a database. At this level, the information conveyed by a diagram is intensional and is comparable to a conceptual schema of the relational model. We will use the term E-R diagram to denote this class of di-

agram. A slightly different type of diagram could be used to represent the extensional information of a database or the database instances. But this type of diagram is not part of Chen's original E-R model.

In an E-R diagram, the entity sets, the relationship sets and the value sets in a database are represented by different type of nodes. The role and attributes are represented by connections between two nodes. Three types of nodes are used, a rectangle for an entity set, a diamond for a relationship set and a circle for a value set. A role that an entity set play in a relationship set is represented by a labeled arc connecting a node representing an entity set to a node representing a relationship set. An attribute is represented by a labeled arrow from a node representing an entity set or a relationship to a node (nodes) representing a value set (Cartesian product of value sets). A general structure of the E-R diagram may be illustrated by the following figure:



Example 2.3
of a simple E-R diagram



2.4 Constraints in the E-R Model.

Constraints are logical properties of data that must be incorporated into a database. These properties reflect the semantics of part of the real world being modeled. The contraints are usually expressed as restrictions on the values of data and/or how the data may or may not be related to one another. A valid database must satisfy all contraints at all time to ensure the semantic integrity of the database. In the E-R model, constraints may be expressed in two different ways. Some constraints may be included as part of an E-R diagram called structural constraints. Additional constraints may be specified as a set of predicates apart from the E-R diagram. We call this type of contraints explicit constraints.

2.4.1 Structural Constraints.

The type of constraints which may be specified as part of the E-R diagram (conceptual schema) are the following.

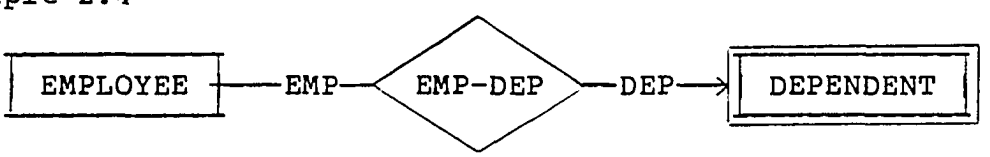
2.4.1.1 Set Membership Constraints.

For every value set Vi or entity set Ei, there exists a primitive predicate Pi that can be used to decide whether a value or entity belongs to the set or not. All entities and values used in a database must satisfy these predicates at all time.

2.4.1.2 Existency Contraints.

This type of constraints requires that for an entity of type E0 to exist, it must be related to the entities of type E1, E2, ..., En. In another word, the existence of an entity in E0 is dependent on the existence of entities in E1, E2, ..., En. The E-R diagram can express the existence dependency by specifying a special type of entity set called weak entity set. A weak entity set is dependent on other entity sets that participate in the same relationship set. We use a double rectangle box to represent a weak entity set and the arc connecting a relationship set to this entity set is changed to an arrow. An example of the existency constraints is shown in the diagram below.

Example 2.4



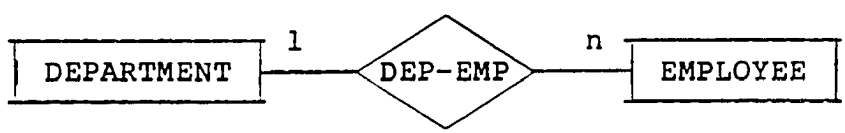
In the E-R diagram shown above, the arrow and the double retangle indicates that the existence of an entity in the entity set DEPENDENT depends on the corresponding entity in the entity set EMPLOYEE. Consequently, if an employee is

deleted from the EMPLOYEE set, then all of his dependents must be deleted also. The relationship set EMP-DEP used to expressed this type of constraint is called a weak relationship set.

2.4.1.3 Connectivity Constraints.

By definition, a relationship set is simply a set of mapping among several entity sets. All possible combination of entities drawn from appropriate entity sets are valid instance of a relationship set. But when we use a relationship set to represent an actual relationship type, we may want to restrict the members of the set to reflect some real world semantic. In the E-R diagram, the mapping property is given explicitly by labeling an arc with the maximum cardinality of an entity set in a relationship set. For a binary relationship we can easily distinguish between one-to-one, one-to-many and many-to-many type mappings.

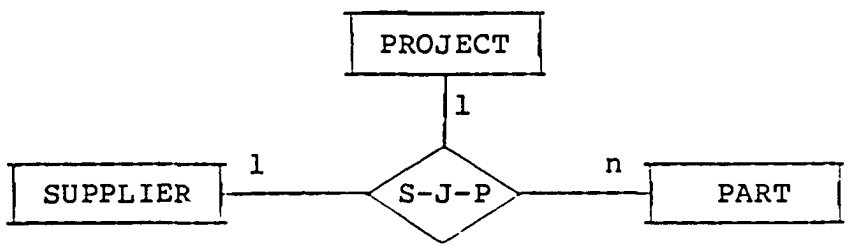
Example 2.5



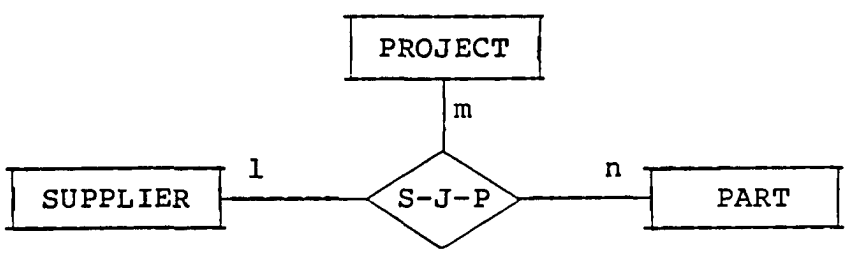
In example 2.5, the relationship set DEP-EMP is a one-to-many mapping as indicated by the labels 1 and n on the two edges. The meaning of this relationship set is one department may have n (0, 1, 2, ...) employees but each employee works for only one department.

The relationship set EMP-PROJ is a many-to-many mapping. That is each employee may work on several projects at the same time and each project may have several employees assigned to it.

The connectivity constraints may be applied to a relationship set involving more than two entity sets. For example, consider a ternary relationship set S-J-P relating the relationship sets, SUPPLIER, PROJECT, and PART.



The diagram shown above indicates that each project is supplied by a unique supplier. And each supplier supplies unique parts.

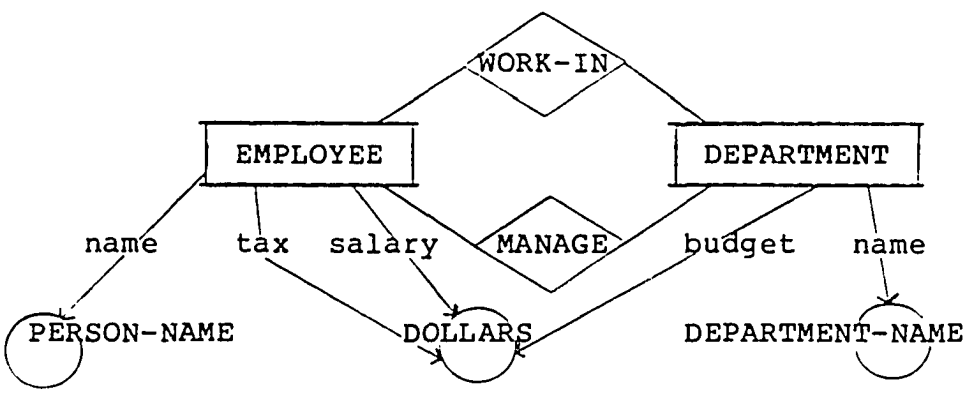


The second S-P-J diagram represents different semantic information. The labels 1, m, n indicates that one supplier supplies several parts to several project. 2.4.2 Explicit Constraints.

Additional constraints which are impossible to include structurally, may be expressed as explicit constraints by the constraint specification facility. Most of the con-

straints in the relational model are of this type. In the E-R model, the explicit constraints are expressed by the first order predicate calculus and aggregate functions such as sum, count, average, minimum, maximum, etc. This type of constraint is a very powerful tool and can be used to expressed almost any kind of semantic information. A few explicit constraints are shown based on a database which is represented by the following E-R diagram. The database includes two relationship sets, WORK-IN and MANAGE, between two entity sets, EMPLOYEE and DEPARTMENT.

Example 2.6



We may express a simple constraint between particular values for example

```
C1)  for every e
       ( is-EMPLOYEE(e) ===> tax(e) <= salary(e) )
```

The constraint C1 states that every employee's tax must be less than his or her salary.

```
C2)  for every ei, for every ej (
       is-EMPLOYEE(ei) and is-DEPARTMENT(ej) and
       is-WORK-IN([ei,ej])
       ===> budget(ej) >= sum(salary(ei)) )
```

The constraint C2 states that a total salaries of every

department may not exceed the department budget.

```
C3)  for every ei, for every ej, there exist ek (
       is-EMPLOYEE(ei) and
       is-EMPLOYEE(ej) and
       is-DEPARTMENT(ek) and
       is-WORK-IN([ei,ek]) and
       is-MANAGE([ej,ek])
       ===> sal(ei) <= sal(ej)    )
```

The constraint C3 indicates that all manager' salaries must be higher than those of their employees.

CHAPTER THREE

ABSTRACTION IN DATABASE MODELS

3.1 Introduction.

A database is a dynamic model of a system that exists in the real world. The system may be an organization such as a city government or an industrial enterprise, or it may be a physical object such as a piece of machinery or an organism. All databases use abstraction in the modeling process. In [S6], abstraction is defined as a model of a system in which certain details are omitted. The users of the model may select the relevant details of the system suitable to their need, and concentrate on a certain part or aspect of the model. Because abstraction allows certain details to be temporarily suppressed a database becomes more comprehensible and more manageable intellectually. All data models in existence use some form of abstraction to a certain extend.

The Relational model [D1] uses abstraction to organize related facts (represented by tuples) into a relation. The result of the process is the relation schema in which all details of individual tuple are suppressed. Only attribute names, which all tuples in the same relation have in common, remain visible at the schema level.

In the IMS (hierarchical) model [K1], a database

record type is an abstraction of database records of the same type. A database record type is a tree of related segment types used as a template for database records.

In the CODASYL (network) data model [O1], a schema which is part of the database definition is an abstraction of the stored database. A schema contains different record types and set types. A record type specifies the logical structure of record occurrences in a database. A set type specifies the connection among record types and represents set occurrences in a database.

Abstraction is used in the Entity-Relationship model to classify entities, relationships and values into different entity sets, relationship sets and value sets. These three set types are the basic building blocks of a database using E-R model.

3.2 Aggregation and Generalization Abstraction.

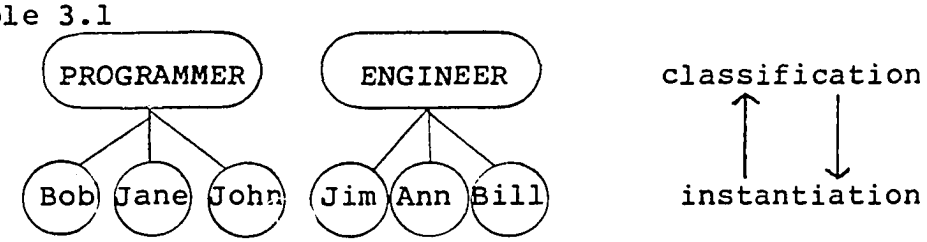
In 1977 and 1979 Smith [S5, S6] introduced two types of abstractions called generalization and aggregation. Each abstraction may be applied repeatedly to generate "objects" at different levels of abstraction. Smith arranged all objects into 2 perpendicular hierarchical structures corresponding to generalization and aggregation abstractions. The two structures are interrelated and can be used to represent rather complex semantic information of the system being modeled. This representation was not possible before the advent of the two abstract structures.

The definition of generalization was given by Tsichritzis [T1] as a process by which a class of individual objects is thought of as a generic object at a higher level. This concept has been further divided into 2 different types of abstraction, e.g., Roussopoulous [R2], Tsichritzis [T1], and Mylopoulous [M2].

3.2.1 Classification-Instantiation.

Classification is an abstraction defined between tokens and a type. A token is an indivisible object in a database used to represent an entity or object in the real world. A type is an abstract object consisting of several tokens as its members. The opposite process of classification is called instantiation. This particular abstraction can be found in most of the data models being used. An example of classification is shown in example 3.1 where a set of individual tokens representing all programmers hired by a company are viewed as the abstract object PROGRAMMER and another set of tokens are viewed as the object ENGINEER.

Example 3.1

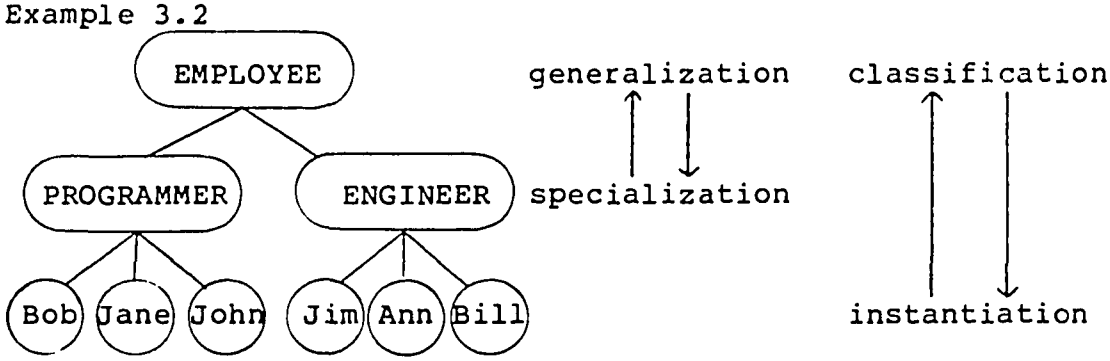


3.2.2 Generalization-Specialization.

Generalization is defined between a set of types and a higher level type and the opposite process is termed spe-

cialization. For instance, viewing the objects PROGRAMMER and ENGINEER as the generic object EMPLOYEE is considered a generalization.

Example 3.2



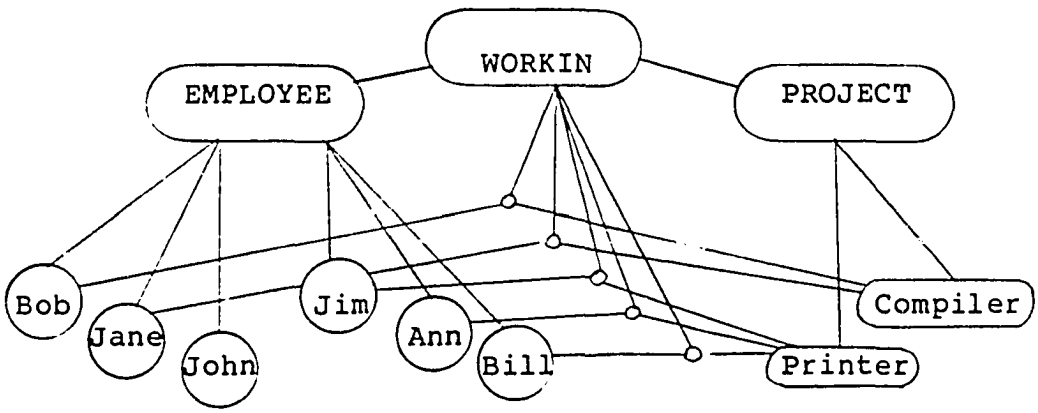
Since EMPLOYEE is a generalization of both PROGRAMMER and ENGINEER, EMPLOYEE includes all members of PROGRAMMER and ENGINEER as its members. We may think of a generic object as a class and its specialized objects as subclasses. This implication applies to all level of the generalization hierarchy.

3.2.3 Aggregation-Stepwise Refinement.

Smith gave the definition of aggregation as a process by which an association between objects is viewed as an aggregate object at a higher level. The counterpart of aggregation in the opposite direction is called stepwise refinement [H4]. This process has been used extensively in programming languages for a long time. An aggregate object at type level always represents a set of aggregate tokens. Thus an aggregate object can be considered as a classification between a type and tokens also. An aggregate object

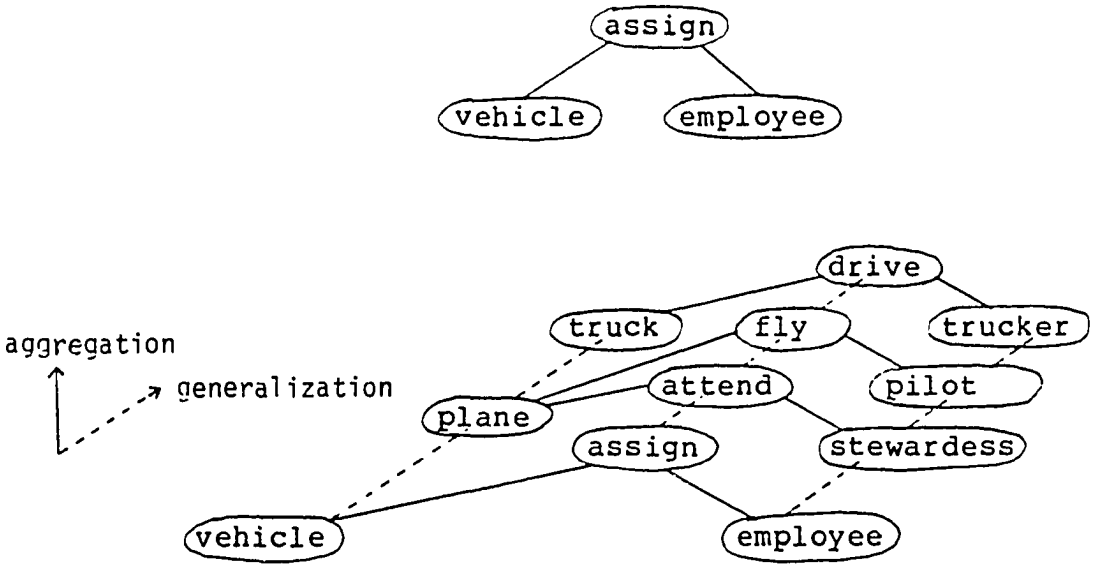
WORKIN is shown in the following example.

Example 3.3



An abstract object may participate in further generalization or aggregation at the higher level. And we may generate abstract structures of arbitrary complexity. The descriptive power of abstract model comes from the connections among the components of the two structures. Smith suggested a graphical representation in which generalization and aggregation are represented orthogonally. Aggregation is drawn on the plane of the page and generalization in the plane perpendicular to the page. For aggregation, high level objects appear toward the top of the page and low level objects toward the bottom. For generalization, high level object appear near the surface of the page and low level objects deep inside the page. The advantages of abstract hierarchies are illustrated by the following figures.

Example 3.4



The top diagram in example 3.4 represents a database of an airline system with one level of abstraction. There are three abstract objects in this diagram. The objects vehicle and employee are generic objects and they represent a group of airline employees and a group of vehicles belonging to the company. The aggregate object (also a generic object) assign represents the set of associations between individual employee and individual vehicle. The token-type level classifications are assumed to exist but not shown here. The second diagram represents the same airline system with multi-level hierarchies. The object employee is now a generic object of 3 objects, stewardess, pilot and trucker. The object vehicle is also specialized into two objects, truck and plane along the generalization plane. The object assign is viewed as a generic object of the objects, attend,

fly and drive. These three object are also aggregate objects in their own right. Again the tokens representing individual entities are omitted to unclutter the diagram. Obviously, the second diagram includes more intricate details in its structures and relationships. Hence the two hierarchies enable us to convey detail semantic information more accurately in a database.

In addition, we may associate new properties with an abstract object in a way that could not be done gracefully before. For instance, a property flight-hour may be associated with the pilot object. This property is type specific and can not be attached to the object employee without introducing non-applicable value for non-flying member of employee.

Generalization and abstraction are used extensively in semantic network in artificial intelligence. The terminologies used are different from the ones given above. The following table give equivalent terms used in database and artificial intelligence.

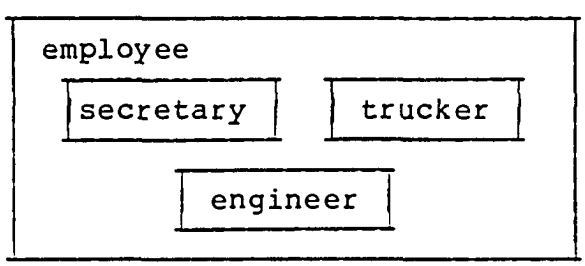
Meaning	database term	artificial intelligence term
association between objects	aggregation stepwise refinement	- part-of
type of tokens	classification instantiation	- instance of
type of types	generalization specialization	- is-a

3.3 Abstraction in the E-R Model.

The generalization and aggregation abstractions first introduced to the relational model have been extended, modified and applied to the E-R model as well. Several articles are cited here together with brief description for each abstract concept.

In [D2], Dos Santos et. al. have investigated existency and operational constraints in the E-R model. They introduce three constructors called sum, product and correspondence in addition to the basic types of entity, relationship and value. The three additional constructors are used to create abstract types from basic types. The sum and product constructors correspond with generalization and aggregation abstractions. A sum is used to create a derived type from a union of several constituent types. The new type is represented by a rectangle drawn around its constituent types.

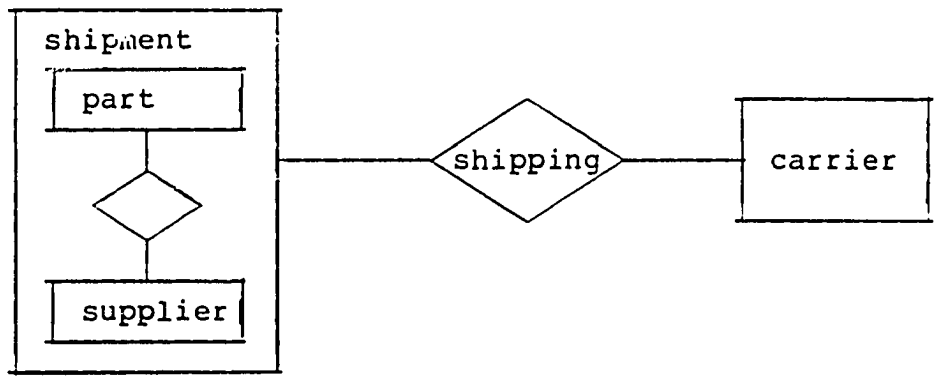
Example 3.5



A product derives a new type from a relationship type and all types participated in that relationship. The derived type is also graphically represented by a rectangle drawn

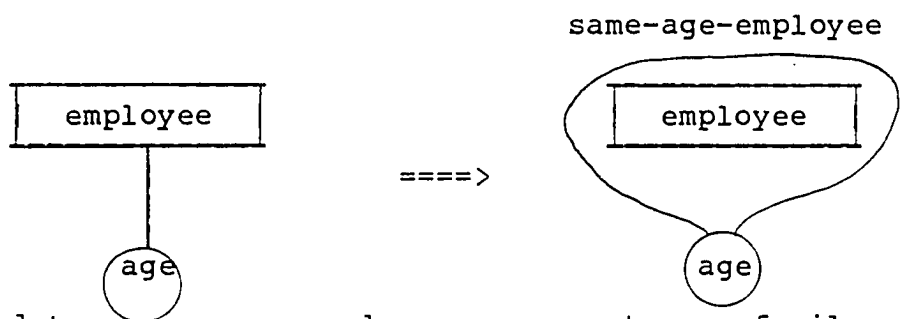
around its constituent types. The following diagram show a product type shipment which can be used to participate in another relationship type.

Example 3.6



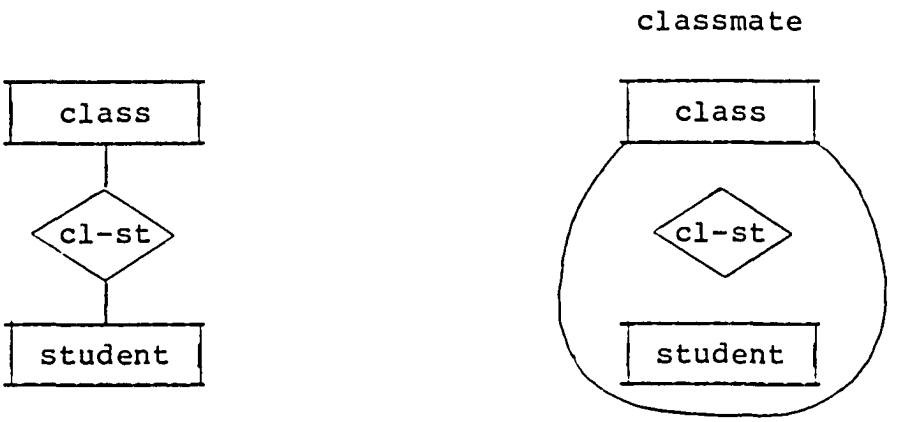
A correspondence constructs a derived type from several types by designating one type as an indexed type and the remaining types as indexing types. The new type is represented by a circle drawn around the indexed type and attached to the indexing types. In usual practice, either a set of value types or a set of related entity types is used as the indexing types. Two examples of correspondence are given below.

Example 3.7



The derived type same-age-employee represents a family of

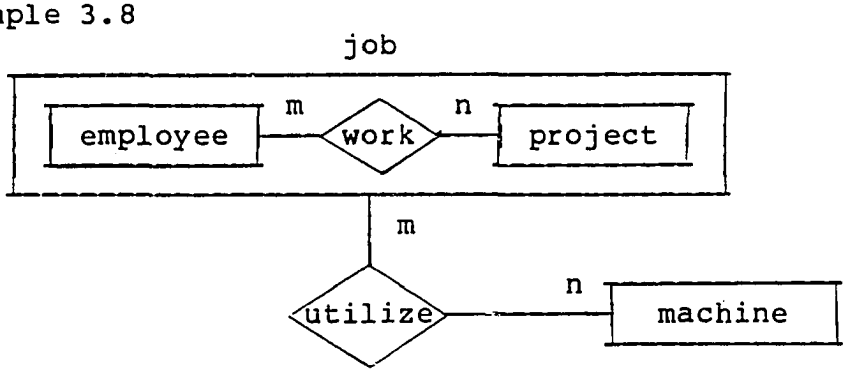
sets of employee whose age are the same.



The derived type classmate has as a member a set of students who belong to the same class. (classmate is a set of sets)

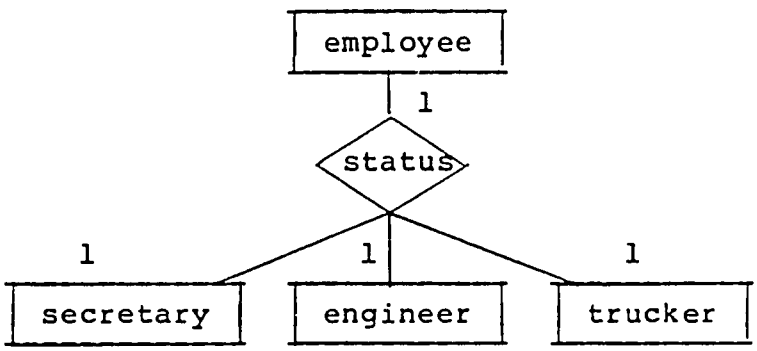
The effects of update operations on E-R databases were studied by Scheuermann et at [S3]. The E-R model has been extended to include aggregation and generalization abstraction. The aggregation turns a relationship set and all entity sets involved into a new entity set. This idea is similar to the product concept mentioned earlier. The following diagram shows that an entity set created by aggregation may participate in a relationship like all other entity sets.

Example 3.8



Generalization is applied in this model as a relationship set that associates a generic entity set to its constituent entity sets. The diagram below shows a generic hierarchy created by this approach.

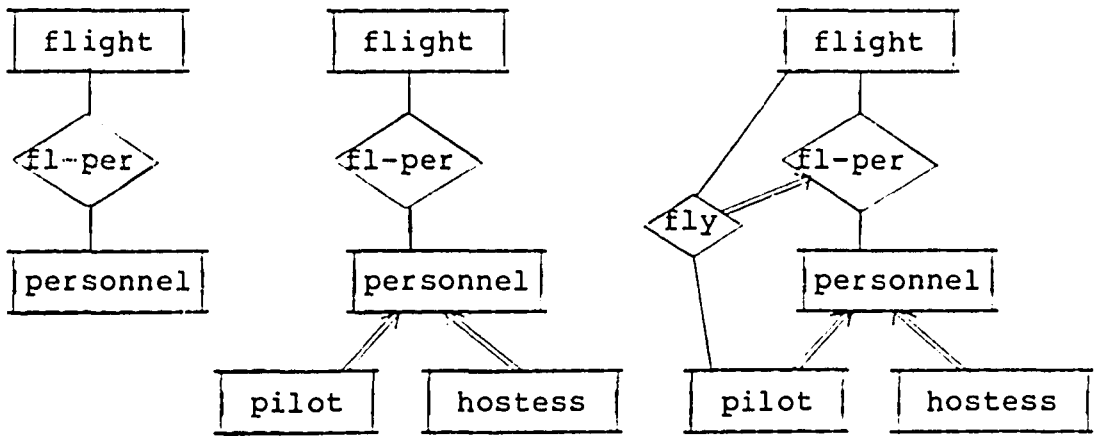
Example 3.9



The authors associated several invariant properties with the structures created by abstractions. They showed that the type of the invariant properties and the type of update operations uniquely determine the way an operation is propagated through out a database.

In the article by Batini and Santucci [B1], the generalization abstraction is used to aid a design technique. A top-down design methodology was studied in the same direction as Chen's top-down design of the E-R model. First, all entity sets and relationship sets are identified. Then, additional semantics are used to refine these sets. The refinement technique involved the generalization of both entity sets and relationship sets. A double arrow is used in a diagram to link an object to its generic object as shown in the following diagram.

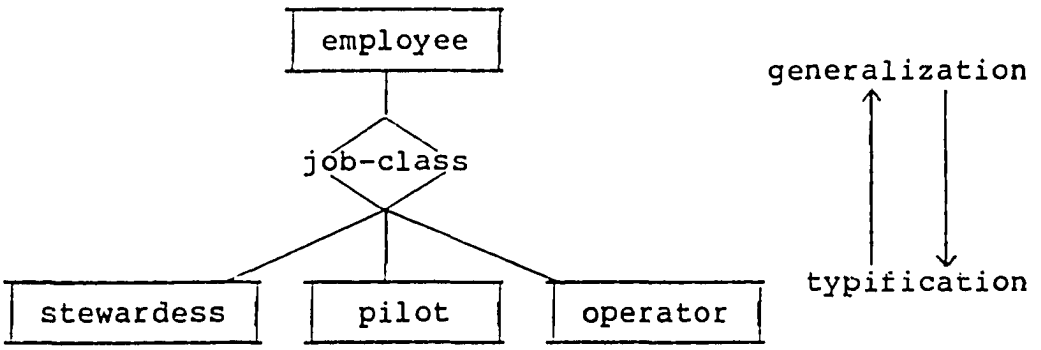
Example 3.10



The left figure in the diagram depicts a database structure after two entity sets and one relationship set have been identified. The middle figure depicts generalization of one of the entity sets. And the right figure shows the generalization of the relationship set.

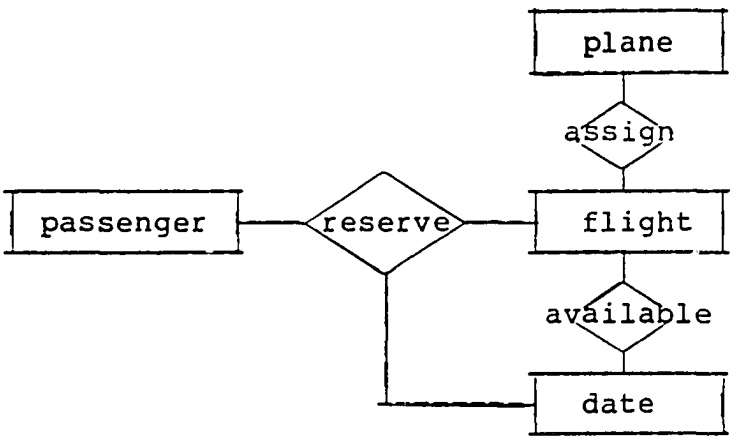
Three types of abstraction were incorporated into the E-R model to improve a normalization process by Sakai [S1]. The resulting schema is used with the transaction descriptions to predict the distribution of transaction workloads on a database. The three types of abstraction include generalization, typification and instatiation. The generalization and typification correspond to Smith's generalization and classification. The instantiation is also related to the term instatiation previously defined. But Sakai used the term instantiation in a very specific sense where a relationship set is turned into an entity set. He claimed that in some situations it is more natural to view a relationship set as an entity set.

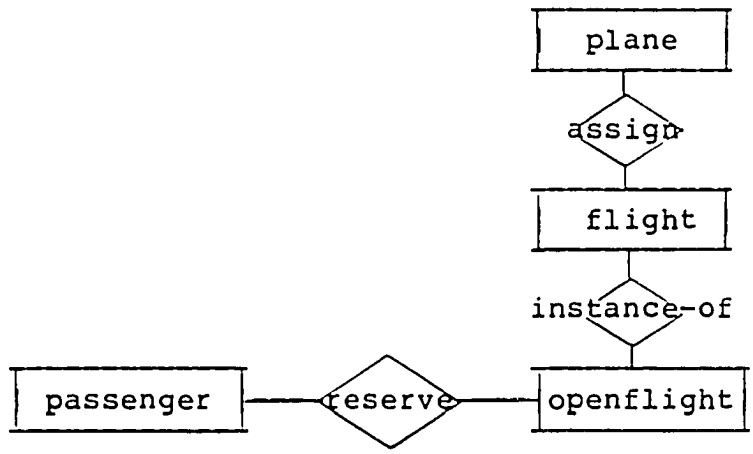
Example 3.11



Generalization and typification are shown in the diagram above where the entity set employee is categorized into the entity sets stewardess, pilot and operator. The category is represented by an oval in the E-R diagram.

Example 3.12





Sakai used the airline database shown as the E-R diagrams above to exemplify the advantage of instantiation. The second diagram shows the result of the instantiation of the relationship set available in the first diagram. The newly created entity set openflight in the second diagram is an instance set of flight by date. The entity set date in the first diagram is assumed to have only one attribute which is the date itself. Since this attribute is also included in the entity set openflight, the date entity set is deleted from the second diagram. The relationship set reserve is also modified to connect to the openflight entity set. This arrangement reflects more naturally the semantics of the airline system under consideration. Sakai also showed that the modified database structure speed up retrieval under certain conditions.

CHAPTER FOUR

AN EXTENDED E-R MODEL

We propose a new model which is an extension of the basic E-R model presented in the first chapter. This model incorporates abstraction concepts which include both generalization and aggregation. Three basic set types of the original E-R model are carried over without change. Entity sets (E-sets) and value sets (V-sets) represent classification abstraction. These two set types associate classes to their tokens. Relationship sets (R-sets) represent aggregation abstraction. Each set represents association among tokens drawn from different E-sets. We add generalization structures to the 3 basic set types creating three generalization structures. Each generalization structure relates generic sets to their specialized sets.

4.1 Generalization Structures.

4.1.1 Generalization Structure of V-sets.

We create a structure on top of V-sets similar to interpreted domains mentioned in [T1]. The structure consists of connections between V-sets, each connection indicates a generalization of one V-set by another. Only compatible values of the same set may be compared in this model. The generalization structure is represented by the pair

VG (Value-Graph) = <FV,GV>, where

FV = { Vi | Vi is a value set },
GV = { (Vi,Vj) | Vi,Vj is in FV }.

The set FV represents a set of all V-sets used in a database. The set GV represents all generic connections between V-sets. For each ordered pair (Vi,Vj) in GV, we say that Vi is a generalization of Vj and Vj a specialization of Vi. The pair <FV,GV> formed a directed acyclic graph which preserves the generalization-specialization abstraction concept.

The rule imposed by this generalization structure is

for each (Vi,Vj) in GV, if v is in Vj ==> v is in Vi.

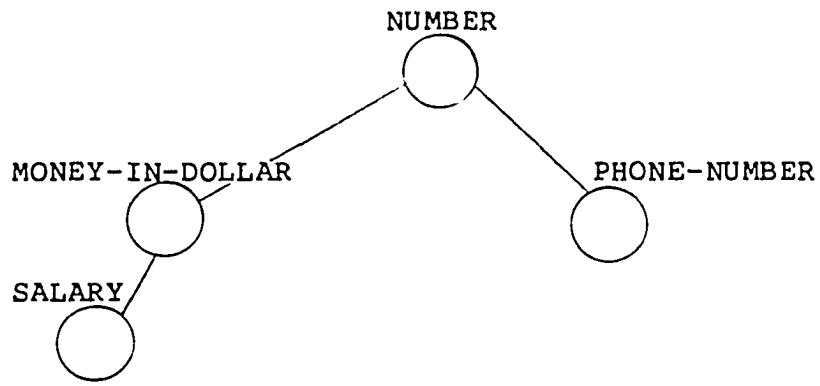
This rule states that all values appearing in a specialized V-set must also appear in its generic V-set. Any intrinsic properties of a value appearing in a generic V-set remain with the value when it appears again in a more specialized V-set.

Example 4.1

This example shows the generic structure that exist among 4 V-sets,

FV = {NUMBER,MONEY-IN-DOLLAR,SALARY,PHONE_NUMBER}
GV = {(NUMBER,MONEY-IN-DOLLAR),
(NUMBER,PHONE-NUMBER),
(MONEY-IN-DOLLAR,SALARY) }.

These two sets may be illustrated by the following diagram:



From this example, MONEY-IN-DOLLAR is a specialized V-set of NUMBER. And SALARY is a specialized V-set of MONEY-IN-DOLLAR. The SALARY V-set inherits the properties from both MONEY-IN-DOLLAR (its parent) and NUMBER (its ancestor). We interpret the properties inheritance as a requirement that each salary is a number and has a basic unit in dollar. At each level of abstraction, additional properties may be attached to a V-set that are not shared by its generic V-sets. For instance, we may add a property that each salary must be between $8000 and $50000 to the SALARY set.

4.1.2 Generalization Structure of E-sets.

The generalization of E-sets has more elaborate structure than the one of V-sets. An E-set may be generalized several times under different categories. The generalization structure is represented by a quadruple

```
EG = <FE,GE,CE,SE>, where
FE = { Ei | Ei is an E-set },
CE = { ci | ci is a category },
GE = { (c,Ei,Ej) | i ≠ j & c in CE & Ei,Ej in FE },
SE = { (Ei,cj,(a,b)) | cj in CE & Ei in FE &
       a in {n,d} & b in {0,1}   }.
```

The set FE represents the family of all E-sets used in a database. The set CE represents all of the category names used in the generalization of E-sets. The set GE contains all generalization connections among E-sets. The triplet (c,Ei,Ej) indicates that Ei is a generalization of Ej and Ej is a specialization of Ei under category c. The set SE represents a set of structural constraints called E-splitting constraints. The triplet (Ei,cj,(a,b)) indicates that specialization of Ei under category cj has the constraint (a,b). The first component of the ordered pair (a,b) indicates the disjointness of all specialized E-sets of Ei under cj. It may be specified as d for disjoint or n for non-disjoint. The second component of the ordered pair indicates the nature of mapping between Ei and all of its specialized E-sets under cj. This component may be specified as 0 for into mapping or 1 for onto mapping. Formally, let

(Ei,cj,(a,b)) in SE and
(cj,Ei,Ek) in GE for k=1,2,...,n.

If a = d ===> $Ek1 \cap Ek2 = \phi$
for k1 $\neq$ k2 and k1,k2=1,2,...n,
otherwise these conditions may not hold.

if b = 1 ===> Ei $\supseteq$ Ek, k=1,2,...,n,
otherwise Ei $\equiv$ Ek, k=1,2,...,n.

The triplet <FE,GE,CE> form a directed labeled graph with nodes drawn from the set FE, labels from the set CE, and arcs from the set GE. This graph has the following pro-

perties.

1) The graph is acyclic to ensure that an E-set can not be a specialization of itself either directly or indirectly.

2) The graph is partitioned into several subgraphs each of which represents a unique concept.

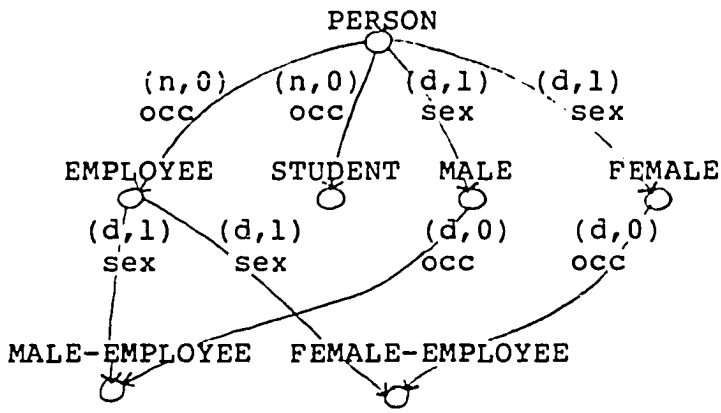
3) Each subgraph has a distinct "root" node whose indegree is zero. This node represents the most general concept in that subgraph.

4) Each subgraph will be called an E-graph and is a part of the extensional information of a database (part of a schema).

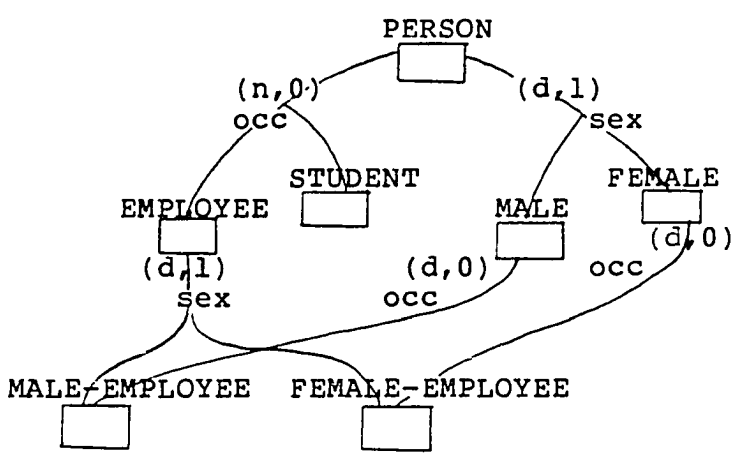
The following graph represents an example of a particular concept in a database.

Example 4.2

```
FE = {  PERSON, EMPLOYEE, STUDENT, MALE, FEMALE
MALE-EMPLOYEE, FEMALE-EMPLOYEE            }

CE = {  occ, sex }
GE = {  (occ,PERSON,EMPLOYEE),
        (occ,PERSON,STUDENT),
        (sex,PERSON,MALE),
        (sex,PESON,FEMALE),
        (sex,EMPLOYEE,MALE-EMPLOYEE),
        (sex,EMPLOYEE,FEMALE-EMPLOYEE),
        (occ,MALE,MALE-EMPLOYEE),
        (occ,FEMALE,FEMALE-EMPLOYEE)   }

SE = {  (PERSON,occ,(n,0)), (PERSON,sex,(d,1)),
        (MALE,occ,(d,0)), (FEMALE,occ,(d,0)),
        (EMPLOYEE,sex,(d,1)) }
```



To conform with the standard E-R diagram, the graph is redrawn by merging arcs under the same node with the same label into a single line with split end. Extraneous labels and constraints are also removed from the graph. And a rectangle is used as a node symbol. The graph shown above may be redrawn as follow.



4.1.2.1 Instantiation of an E-graph.

By using an E-graph as a template and its E-splitting constraints as guidelines, a subgraph of the E-graph may be

created to represent an instantiation of that E-graph. This instantiation is called an e-graph. It represents the intensional information of the concept represented by the corresponding E-graph (database instance). For a given E-graph its e-graph is a subgraph

eG = <FE',CE',GE'> where FE'$\subseteq$FE & CE'$\subseteq$CE & GE'$\subseteq$GE.

4.1.2.2 Definitions of Terms.

In the description that will follow, several relationships among nodes in a graph are pertinent. Different terms are defined to represent these relationships. Some of them are standard definitions from graph theory [S1, H3].

Given a directed labeled graph G=<F,C,G>, where
F = { ai | ai is a node }
C = { ci | ci is a label }
G = { (ci,aj,ak) | ci in C and aj,ak in F }.

A path from a1 to an+1 is a sequence
(c1,a1,a2),(c2,a2,a3),..., (cn,an,an+1), such that
(ci,ai,ai+1) in G for i=1,..,n.

An immediate predecessor of aj is a node ai, such that
(ck,ai,aj) in G.

A predecessor of aj is a node ai, such that
there is a path from aj to ai.

An immediate successor of ai is a node aj, such that
(ck,ai,aj) in G.

A successor of ai is a node aj, such that
there is a path from ai to aj.

A sibling of ai is a node aj, such that
(ck,al,ai),(ck,al,aj) in G for some ck in C

and some al in F.

A cousin of ai is a node aj, such that (cl,am,ai), (ck,am,aj) in G for some cl,ck in C and some am in F.

An nth cousin of ai is a node aj, such that there is a node ak who is a cousin of ai and there is a path from ak to aj.

4.1.2.3 Instantiation Rules.

The following rules create an e-graph from an E-graph and its E-splitting contraints.

1) Include a root node.

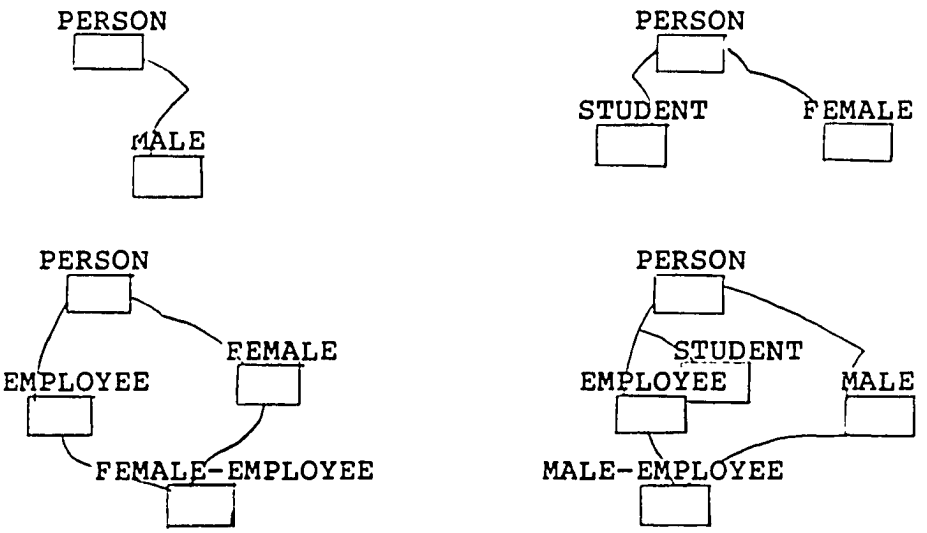
2) Choose any immediate descendant of a node already included in the e-graph. All paths from the root to that node must be included also.

3) If the E-splitting constraint of an immediate predecessor of the node chosen specifies disjointness of successors and a sibling of that node is already included, then that node can not be included.

4) If the E-splitting constraint of any category under the node chosen specifies onto mapping, then at least one of its immediate successors under that category must be included.

The following example illustrated four different valid e-graphs generated from the E-graph given in the previous example.

Example 4.3



4.1.2.4 Semantic Implication of E-graph and e-graph Structures.

The interpretation of an e-graph in the extended E-R model is based on the closed world assumption [R1, G1]. According to this assumption, an e-graph contain the fact about one entity playing different roles represented by different nodes in the e-graph. For example, the first e-graph in the example above represents a male person who is neither an employee nor a student. We also disallow extending an existing e-graph by mean of update operation. Adding more nodes and arcs to an e-graph in a database would require a much more complicated algorithms than the ones which will be presented later on.

Similar to V-sets structure, the rule imposes by the generalization of E-sets is

for each (c,Ei,Ej) in GE, if e is in Ej => e is in Ei.

This rule states that all entities in an E-set must also belong to all generic E-sets of that E-set. Or an E-set must be a subset of all of its generic E-sets. Now associated to each generic E-set, we have set-member association denoting classification-instantiation abstraction and we also have set-subset association denoting generalization-specialization abstraction. The later is optional and several sets at the lowest level in the generalization structure may not have this association.

Example 4.4

This example shows the E-sets ENGINEER, SECRETARY and TRUCKER as specialization of the E-set EMPLOYEE.

```
FE = { EMPLOYEE,ENGINEER,TRUCKER,SECRETARY }
CE = { type }
GE = { (type,EMPLOYEE,ENGINEER),
       (type,EMPLOYEE,TRUCKER),
       (type,EMPLOYEE,SECRETARY) }
SE = { (type,EMPLOYEE,(n,0)) }
```

The same information is shown in the first diagram using modified E-graph. All the instances of this E-sets are also shown in two distinct styles. In the third diagram, the instances are shown as a collection of e-graphs. In the second diagram, entities are represented by unique tokens contained in different circles. Each circle represents an E-set the tokens belong to. Both types of instantiation representation are easily interchangeable.

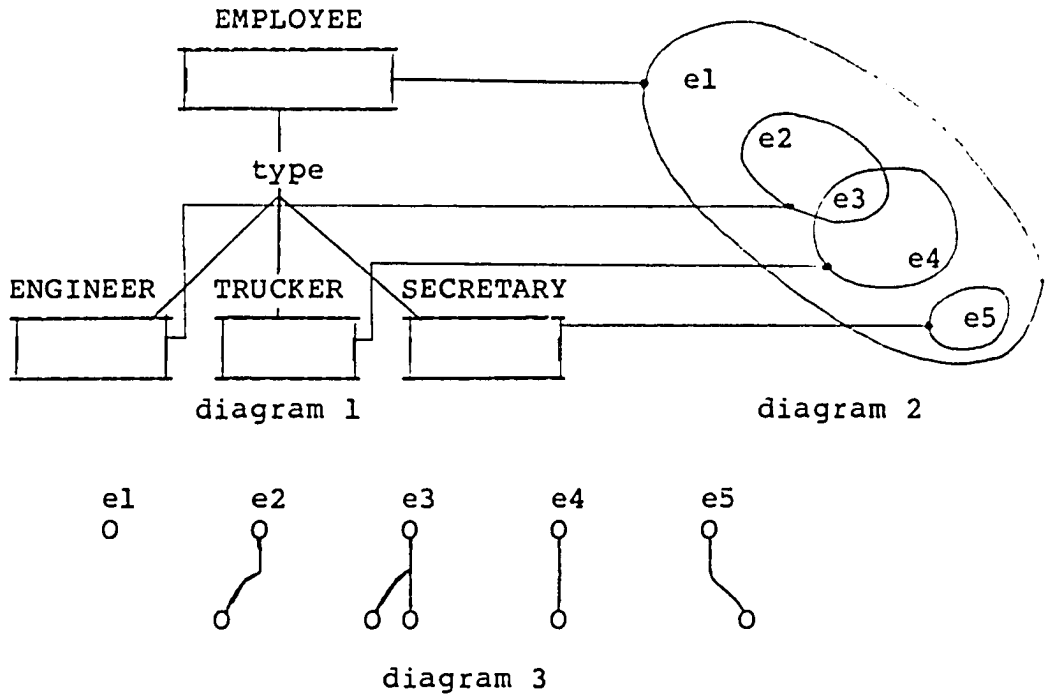
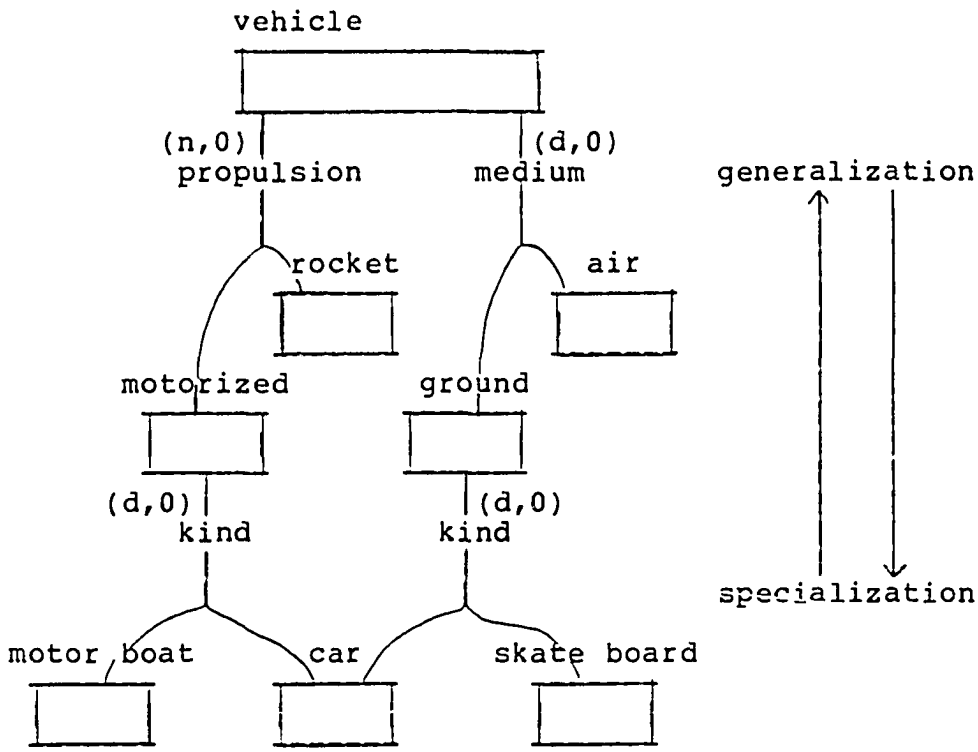


diagram 3

Example 4.5

This example shows a database that involves 3 levels of generalization structure composed from different E-sets. The database is presented as a diagram only. The sets FE, CE, GE, SE and the instantiation of each E-set are omitted. The diagram illustrates that categories under different generic E-sets may be the same, e.g., kind appears under 2 different E-sets. Furthermore, an E-set may have more than one generic E-sets, e.g., car belongs to both motorized and ground E-sets.



4.1.3 Generalization Structure of R-sets.

The concept of an R-set represents both aggregation and classification abstraction in the same structure like the original E-R model. In the extended model, we again add the generalization structure among R-sets. We represent this structure by the quadruple

RG = <FR,CR,GR,SR>, where

FR = { Ri | ri is an R-set },
CR = { ci | ci is a category of R-set specialization },
GR = { (c,Ri,Rj) | i ≠ j & c in CR & Ri,Rj in FR },
SR = { (Ri,cj,(a,b)) | Ri in FR & cj in CR and
a in {n,d} & b in {0,1} }.

The first component FR of this quadruple is a family of all R-sets used in a database. The second component CR is a set

of strings used as category names for R-sets specialization. The third component GR represents all generic connections among the R-sets. The last component SR represents a set of structural constraints called R-splitting constraints each of which is a triplet (Ri,cj,(a,b)). Similar to E-splitting constraint, the last component of the triplet specifies the disjointness of Ri's successors and the mapping characteristic between Ri and the union of it's successors under category cj. Formally, let

(ck,Rj,Ri) in RG for i=1,2,...,n and (Rj,ck,(a,b)) in SR.

If a = d ===> $Ri1 \cap Ri2 = \phi$ for $i1 \neq i2$
and i1,i2=1,2,...,n,
otherwise these conditions may not hold.

If b = 0 ===> a relationship r is allowed to be
a member of Rj even though it cannot be
a member of Ri 1<=i<=n.
If b = 1 ===> a relationship r is not allowed to be
a member of Rj unless it is a member of
some Ri 1<=i<=n.

More detail of E-splitting and R-splitting constraints will be given in the next section.

Similar to the generalization of E-sets, the first three components of RG <FR,CR,GR> form a directed labeled graph whose nodes are drawn from FR, labels from CR and arcs from GR. The graph is part of the extensional information of a database (schema). It has the following properties:

1) The graph is acyclic to prevent an R-set becoming a specialization of itself.

2) The graph is partitioned into several subgraphs each one represents a unique concept of mutually related E-sets.

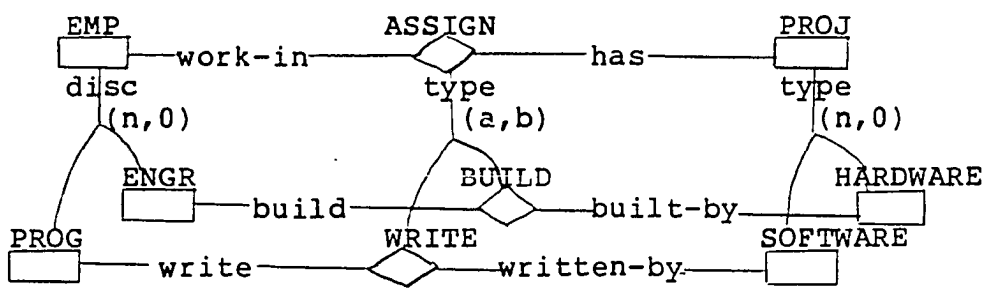
3) Each subgraph will be called an R-graph; each represents a relationship concept.

4) Each subgraph has a distinct "root" node whose indegree is 0. This node is the most general relationship in a unique concept.

The following example show two different E-graphs and an R-graph that relates these two E-graphs.

Example 4.6

```
FE = { EMP,PROG,ENGR,PROJ,HARDWARE,SOFTWARE }
CE = { type,disc }
GE = { (disc,EMP,PROG), (disc,EMP,ENGR),
       (type,PROJ,HARDWARE), (PROJ,SOFTWARE)  }
SE = { (EMP,disc,(n,0)), (PROJ,type,(n,0)) }

FR = { ASSIGN,WRITE,BUILD }
CR = { type }
GR = { (type,ASSIGN,WRITE), (type,ASSIGN,BUILD) }
SR = { (ASSIGN,type,(a,b)) }

where
ASSIGN = (work-in/EMPLOYEE,has/PROJ)
WRITE  = (write/PROG,written-by/SOFTWARE)
BUILD  = (build/ENGR,built-by/HARDWARE)
```



4.1.3.1 Instantiation of R-graph.

Similar to E-graph instantiation, the R-graph is used as a template, and R-splitting constraints are used as rules to follow in the instantiation process. The result is an

r-graph whose structure is a subset of R-graph structure. However, each nodes of an r-graph is connected to nodes of different e-graphs in a database. Thus all e-graphs and r-graphs make up the major portion of the database contents. Formally, each r-graph is a triplet

$$rG = \langle FR', CR', GR' \rangle \text{ where } FR' \subseteq FR \ \& \ CR' \subseteq CR \ \& \ GR' \subseteq GR.$$

4.1.3.2 Instantiation Rules.

An r-graph is created by materialization of different R-nodes in an R-graph. Materialization of an R-node is done by checking all E-nodes connected to that R-node. If each E-node has a corresponding e-node (with some desired properties) in a database, then create a new r-node and connect it to all e-nodes selected. If some of the corresponding e-nodes can not be located, then the materialization fail.

4.1.3.3 Creating an r-graph from an R-graph.

1) Materialize a root node.

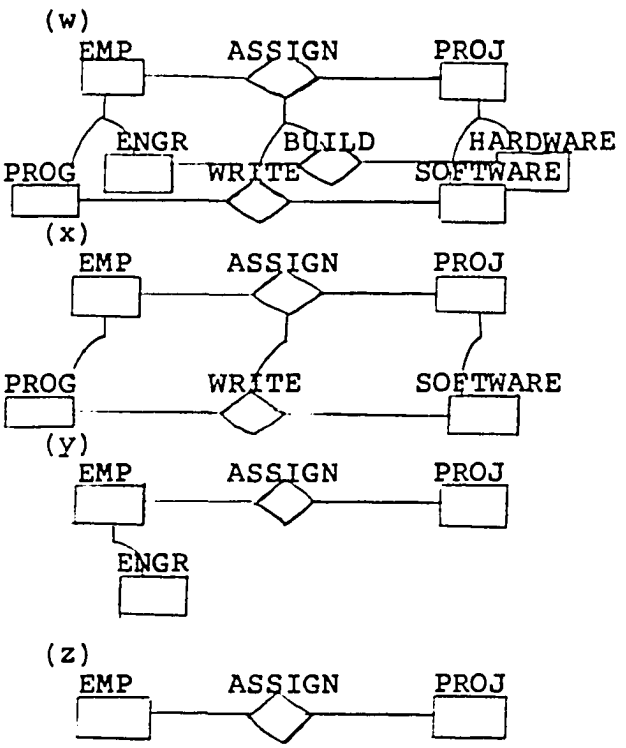
2) For each node materialized, try to materialized all of its immediate successors. If a new node is materialized, then copy the labeled arc from the old node to the new node.

3) Only one of the sibling R-nodes may be materialized if they are disjoint.

4) If none of the sibling R-nodes can be materialized and the R-splitting constraint of their immediate predecessor indicates onto mapping, then the entire r-graph can not be materialized.

Example of valid r-graphs instantiated from the R-graph from the previous example.

Example 4.7



From the instantiation rules used to creating an r-graph, a relationship (r-node) in an r-graph must be propagated downward from the root node through each category down to the leaves. The only time that an r-node may not be propagated through a category under it is when a node can not be materialized due to the absence of appropriate e-nodes in the database and the mapping contraint of that particular category is onto. There are several rules imposed by the generalization structure of R-sets. The first one concerned with set membership is

```
for every (c,Ri,Rj) in GR,
   if r is in Rj => r is also in Ri.
```

This rule states that a specialized R-set must be a subset of its generic R-set(s). The second rule which is different from the generalization of E-sets and V-sets is

```
for every (c,Ri,Rj) in GR,
let Ri represents an association among Eil,..., Ein
and Rj represents an association among Ejl,..., Ejm,
then n=m and
for 1<=k<=n ,either Eik = Ejk or (c,Eik,Ejk) is in GE.
```

The second rule indicates that if Rj is a result of a specialization of Ri under category c, then both Ri and Rj represent associations among the same number of E-sets. In addition, the rule also indicates that for each E-set of Ri there is a corresponding E-set of Rj. And these corresponding sets are either the same or the one belonging to Rj is a specialization of the one belonging to Ri. The third rule limits the specialization to the ones where at least one of the corresponding E-sets is the result of specialization of an E-set. This rule has the additional requirement that

```
for every (c,Ri,Rj) in GR,
let Ri represents an association among Eil,..., Ein
and Rj represents an association among Ejl,..., Ejn
then there exists k,
1<=k<=n such that (c,Eik,Ejk) is in GE.
```

From the third rule, We say that Rj is induced from Ri by Ejk. A specialization of an R-set may be induced by more that one E-sets. In the previous example shown above, the R-sets WRITE and BUILD are induced from R-set ASSIGN by the E-sets EMP and PROJ.

4.2 Attributes and Generalization Structures.

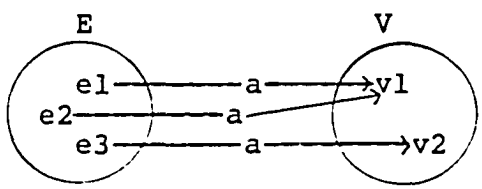
Originally, an attribute is a function from an E-set (R-set) to a V-set or a Cartesian product of V-sets and may be written as

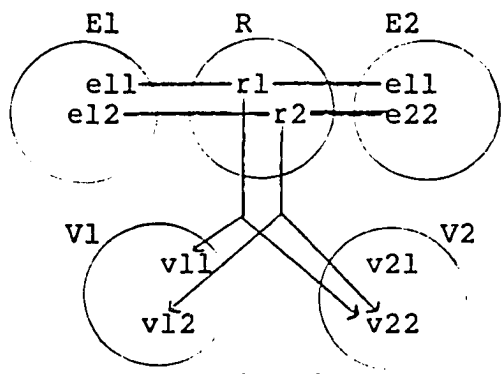
a : Ei or Ri ----> Vj or Vj1 X Vj2 X ... X Vjm

where a is in ATTR (a set of all attributes).

Example 4.8

This concept may be illustrated by the following diagrams:





4.2.1 Property Inheritance Rule for Generalization Structures.

The properties (attributes) of a generic E-set (R-set) are inherited by its specialized E-sets (R-sets). We may put it another way: all common attributes of all specialized E-sets (R-sets) must also belong to their immediate generic E-set (R-set). Formally,

for every (a,Ei,Ej) in GE
if a:Ei -----> Vk or Vk1 X Vk2 X ... X Vkm
then a:Ej -----> Vk or Vk1 X Vk2 X ... X Vkm.

And for every (a,Ri,Rj) in GR
if a:Ri -----> Vk or Vk1 X Vk2 X ... X Vkm
then a:Rj -----> Vk or Vk1 X Vk2 X ... X Vkm.

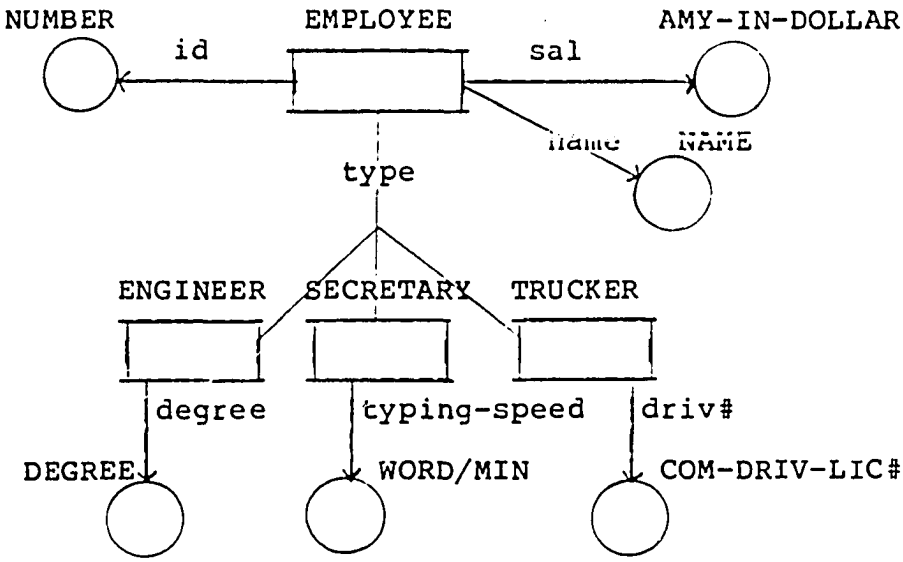
An example of the inheritance of properties are shown using

the database defined in example 3.2 with additional attributes associated with different E-sets as stated below.

Example 4.9

```
ATTR = { id,name,sal,degree,typing-speed,driv# }

id:EMPLOYEE  ----> NUMBER
id:TRUCKER   ----> NUMBER
id:ENGINEER  ----> NUMBER
id:SECRETARY ----> NUMBER

name:EMPLOYEE  ----> NAME
name:TRUCKER   ----> NAME
name:ENGINEER  ----> NAME
name:SECRETARY ----> NAME

sal:EMPLOYEE  ----> AMT-IN-DOLLAR
sal:TRUCKER   ----> AMT-IN-DOLLAR
sal:ENGINEER  ----> AMT-IN-DOLLAR
sal:SECRETARY ----> AMT-IN-DOLLAR

degree:ENGINEER ----> DEGREE
typing-speed:SECRETARY ----> WORD/MIN
driv#:TRUCKER ----> COM-DRIV-LIC#
```

As these functions shown, all common properties of the specialized E-sets (SECRETARY, TRUCKER, ENGINEER) also belong to their generic E-set (EMPLOYEE). But a specialized E-set may have its own attributes not shared by its generic E-set. For instance, the property typing-speed belongs to the set SECRETARY only. The attributes presented may by show as an E-R diagram as follow.



In this diagram, all attributes of the generic E-set EMPLOY-EE belong to each specialized E-set (ENGINEER, SECRETARY and TRUCKER) implicitly through the generalization structure.

4.2.2 Type Specific Attributes.

In the extended E-R model, we include a new type of attributes called type specific attributes. An attribute of this type is a mapping from an E-set or R-set to a value drawn from a V-set. And it can not be inherited through generalization structures. We define a type specific attribute to be an ordered pair

a = (Ei,v) where Ei is in FE and v is in Vj.

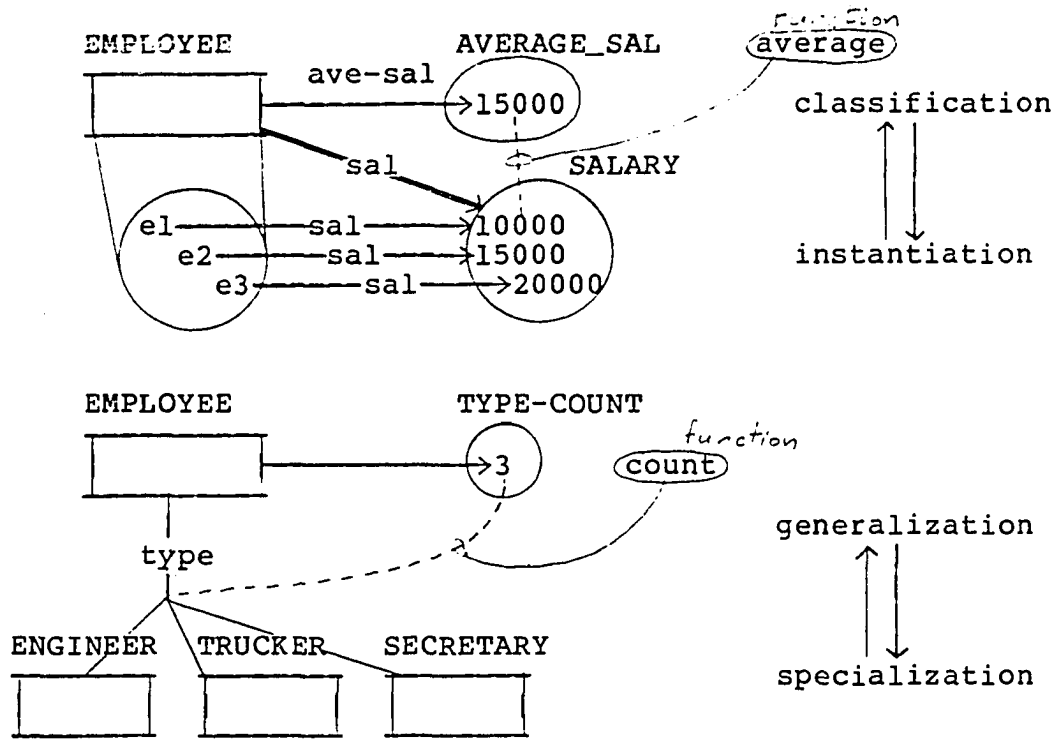
Or a = (Ri,v) where Ri is in RE and v is in Vj.

The first class of this type of attribute is related to aggregate functions. A type specific attribute of an E-set (R-set) may be related to other attributes of the same set

or some other sets. This relationship usually involve some aggregate functions like sum, average, count, maximum, minimum, etc. In this case, the V-set related to an E-set (R-set) through a type specific attribute can be considered a virtual V-set where its value may be derived by appling some aggregate function on existing values or connections.

Example 4.10

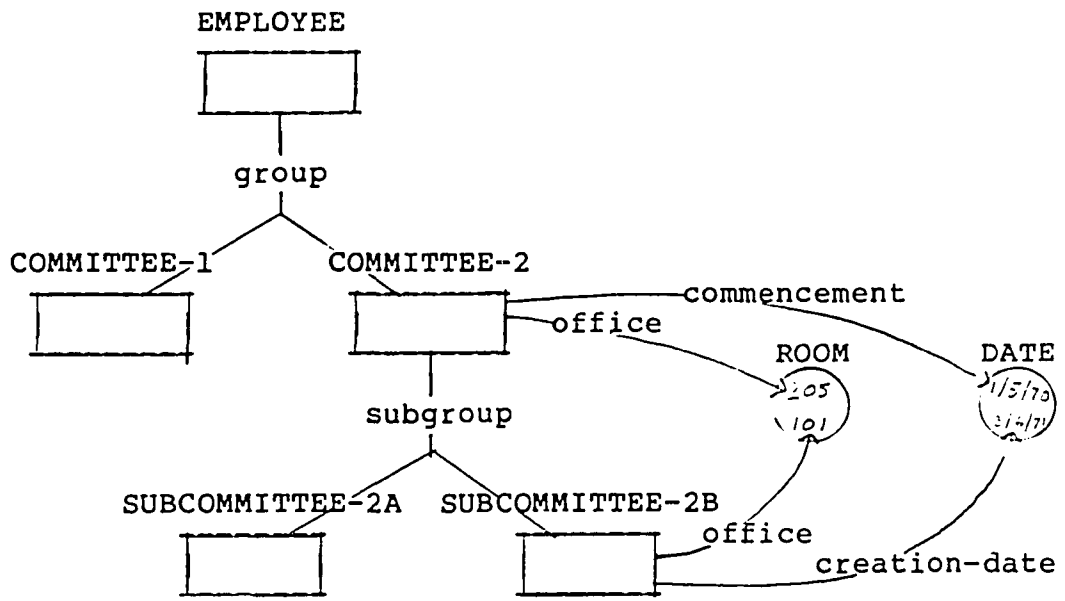
This example shows two different type specific attributes. In the first diagram, ave-sal relates an E-set to a virtual value set based on another value set. On the other hand, in the secon diagram, count relates an E-set to a virtual value set based on the connections in a generalization structure.



The second class of type specific attribute does not base

its values on existing value sets or generic connections. Several attributes in this class are shown in the following example.

Example 4.11



All attributes shown in this example are type specific. Even though the attribute name office appears twice for the set COMMITTEE-2 and SUBCOMMITEE-2B, they are not the same attribute inherited through the generalization structure. They are two different attributes. This point is very important. If the second (bottom) office is inherited, then the two attributes represent the same information. Since both offices are type specific attributes, they represent two different pieces of information.

4.2.3 Semantic Implication of Type Specific Attributes.

A type specific attribute is a connection between an E-set (R-set) and a single value drawn from some V-set; not a set of connections from individual members to some set of values. This definition implies that a type specific attribute represents both extensional and intensional information [T1] or represents both factual and definitional properties [M2]. As an example, consider the 2 previous examples, the attribute ave-sal of EMPLOYEE is regarded as a property of all employees currently hired and not of any individual employee. The attribute commencement of COMMITTEE-2 is also regarded as a property of the set itself and not a definition of actual properties of the individuals in the set.

This interpretation, while intuitively simple, is a major departure from the E-R model semantic interpretation. Now the different sets viewed by the E-R model as intensional information carriers may be viewed by the extended model as carriers for both intensional and extensional information. By relaxing the distinction between definition and instance, the extended model can be made more flexible. It can accommodate more complex semantic information with ease. Even though the 2 types of information are allowed to mix in the extended model, the major structures of the E-R model are kept intact. The strength of the original E-R model namely simplicity, and clarity comes from these basic set types.

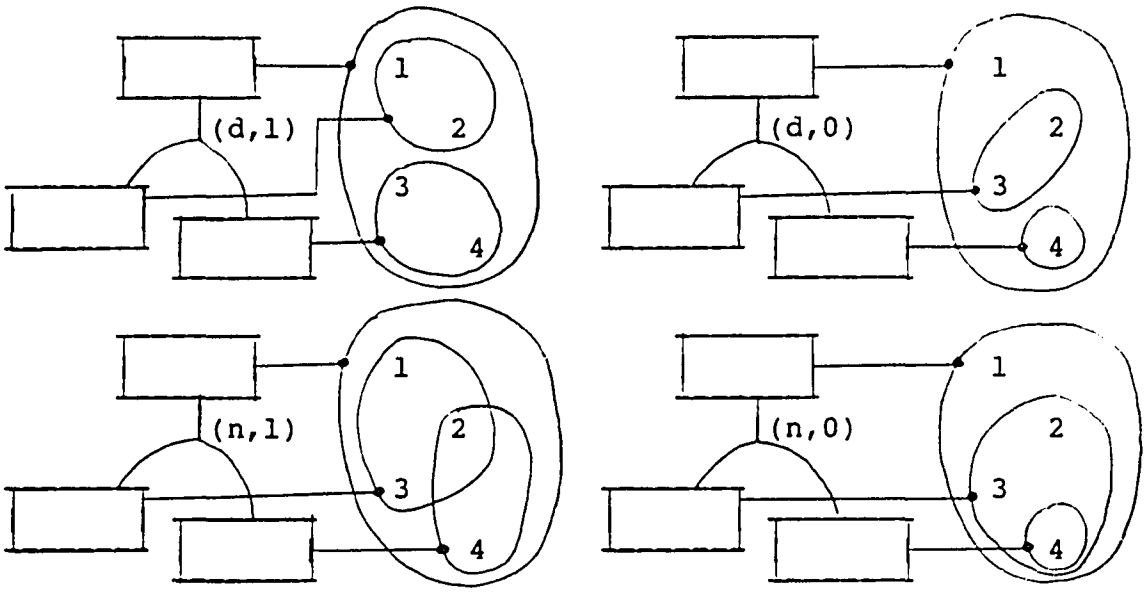
4.3 Structural Constraints of the Extended E-R Model.

The extended model inherits all types of structural constraints from the basic E-R model. In addition, two types of structural constraints called E-splitting constraints and R-splitting constraints are included to specify the characteristic of the generalization structures of E-sets and R-sets.

4.3.1 E-splitting Constraint.

There is an E-splitting constraint associated with every category under an E-set in our model. Each E-splitting constraint consists of two components. One is called a disjoint constraint and the other a mapping constraint. A disjoint constraint specifies whether the sibling E-sets under a category may share common entities or not. A mapping constraint indicates whether or not every entity appearing in the generic E-set must also appear in at least one of its successors under a category. Example of different E-splitting constraints are shown in the following four diagrams. Each diagram shows three different E-sets involved in a single generic structure. The effect of the E-splitting constraints are shown as the correspondences between the E-sets and their instantiations. The notation (a,b) below signifies the splitting constraints.

Example 4.12



4.3.2 R-splitting Constraint.

An R-splitting constraint is associated with each category under an R-set. This constraint defines an association between a generic R-set and all of its specialized R-sets under one category. Each R-splitting constraint consist of two components similar to the E-splitting counterparts. The combination of the two components dictates whether or not a relationship is allowed to exist in the generic R-set without having to appear in at least one or exactly one of the specialized R-set under a category.

Given an R-set R relating a set of E-sets Ei's, where i=1,...,m. Assume that a specialization of R under category c into a set of R-sets Rj, where j=1,2,...n, is induced by E1,E2, ... ,El for some l <= m. The possible number of Rj's (specialized R-sets of R) depends on the number of special-

ized E-sets of each Ei, i=1,2,...,l.

We define the specialized R-set Rj as an R-set relating m E-sets $Ej_1, Ej_2, \ldots, Ej_m$, where

$(c1, E1, Ej_1)$ is in GE,

$(c2, E2, Ej_2)$ is in GE,

.
.
.

(cl,El,Ejl) is in GE, and

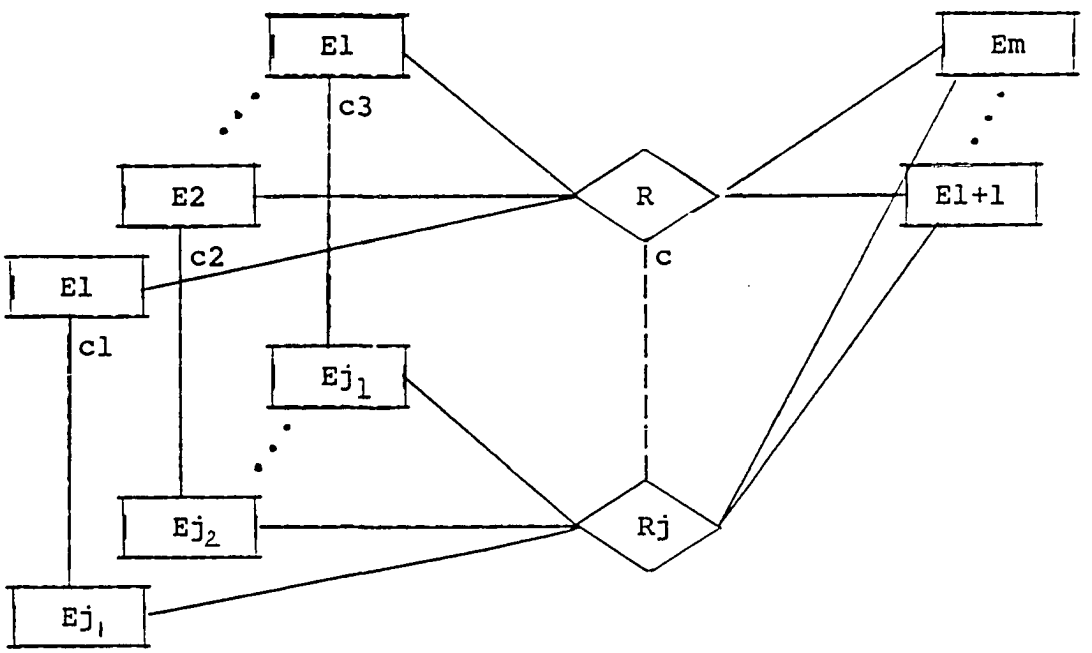
El+1 is Ej_{l+1},

.
.
.

Em is Ej_m.

For some i, 1<=i<=l, Ei_1, Ei_2, ..., Ei_m are not necessarily distinct.

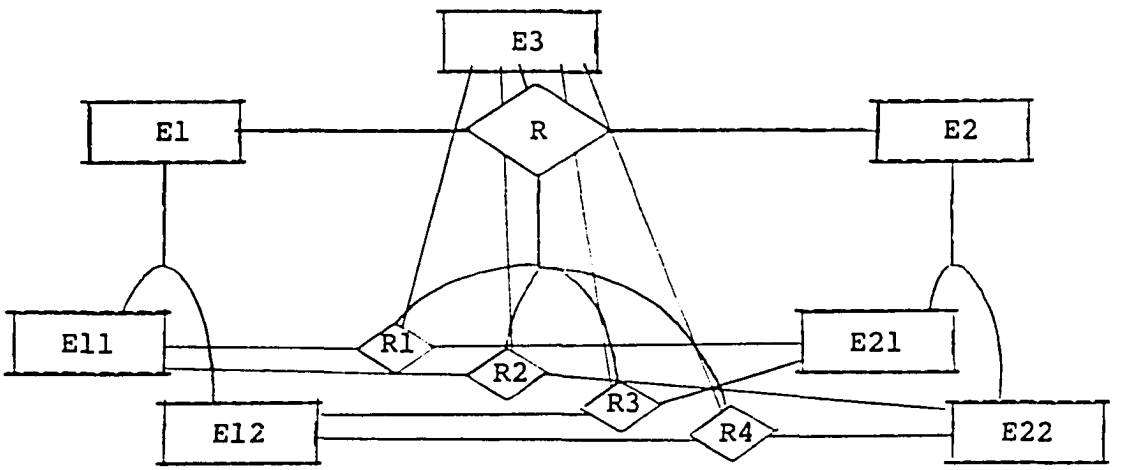
We may use the following diagram to illustrate this relationship.

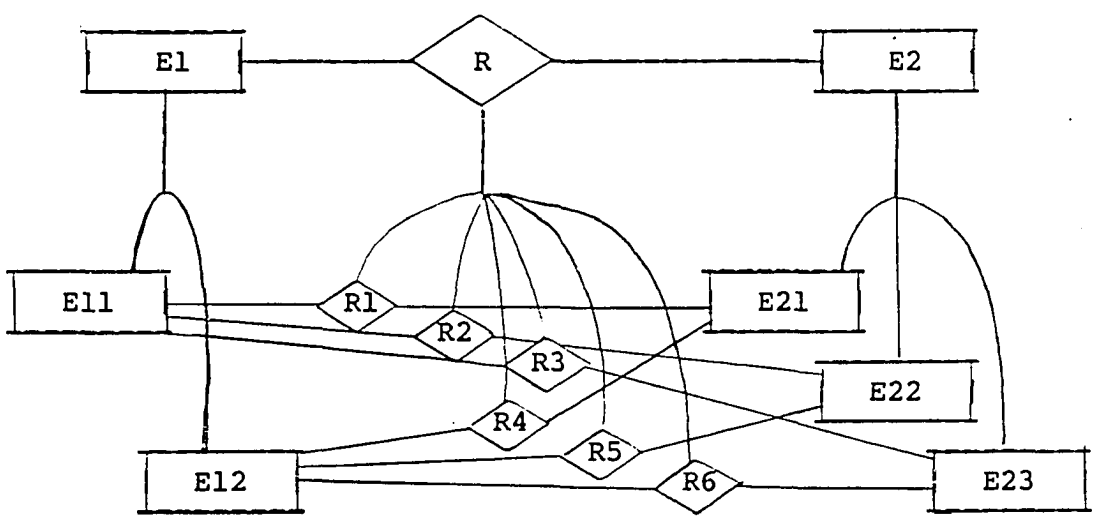


An example below illustrates the possible number of specialized R-sets. The first diagram shows that four specialized

R-sets are possible when each of the two inducing E-sets has two specialized E-sets. The second diagram shows six specialized R-sets are possible when the inducing E-sets have two and three successors respectively.

Example 4.13

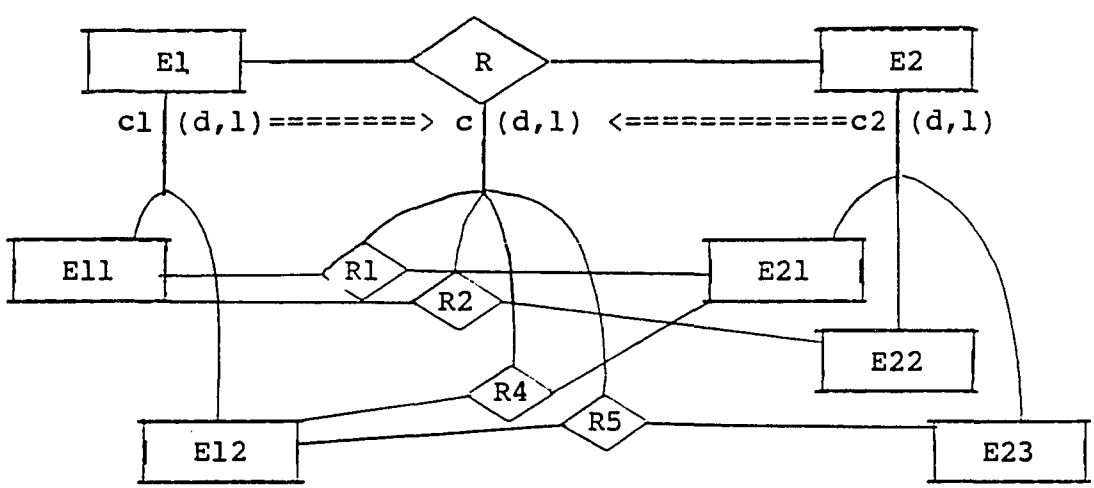




Even though the number of possible specialized R-sets is usually large, only a few may be selected. The selection depends on the semantics of the system being modeled. Out

of all possible combinations only a few may be feasible semantically. It is also possible that we may choose to include in a database only these meaningful R-sets in which we are interested at this point in time. For example, the second diagram above shows a specialization of R induced by the specializations of E1 and E2. We have shown that there are 6 possible combination of the specialized E-sets. Assuming that R1, R2, R3, R4 and R5 are meaningful according to some semantic information discovered by the designer. In this case, R6 is always an empty R-set and can be eliminated from a schema. Out of the remaining R-sets we may eliminate R3 simply because we are not interested in it. Thus, the specialization of R in a database now consists of 4 specialized R-sets, R1, R2, R4 and R5. This specialization is shown in the following example.

Example 4.14

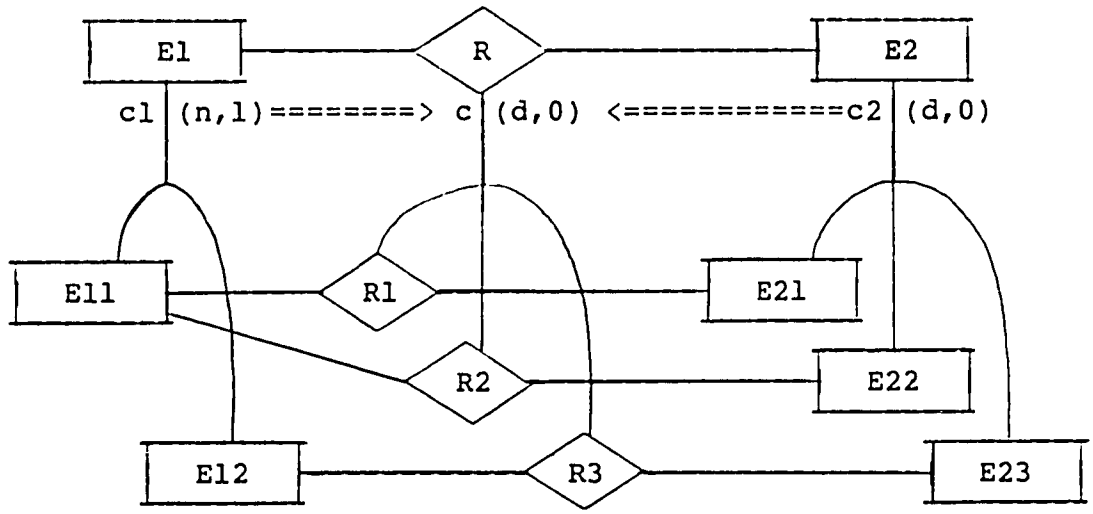


4.3.3 Relationship Between E-splitting and R-splitting Constraints.

Because the specialization of an R-set in the extended model is always induced by the specializations of some E-sets, the R-splitting constraint is always influenced by the E-splitting constraints of the inducing E-sets. Let R be an R-set related to a set of E-sets Ei's, i=1,...,m. And let Rj's, j=1,...,n, be immediate successors of R under category c. Assume that c of R is induced by the categories c1 of E1, c2 of E2, ..., and c1 of E1 for some 1 <= m. If the disjoint constraints of c1,...,c1 all indicate disjointness, then the disjoint constraint of c must indicate disjointness also. An example of this case is shown in example 4.3.3 above.

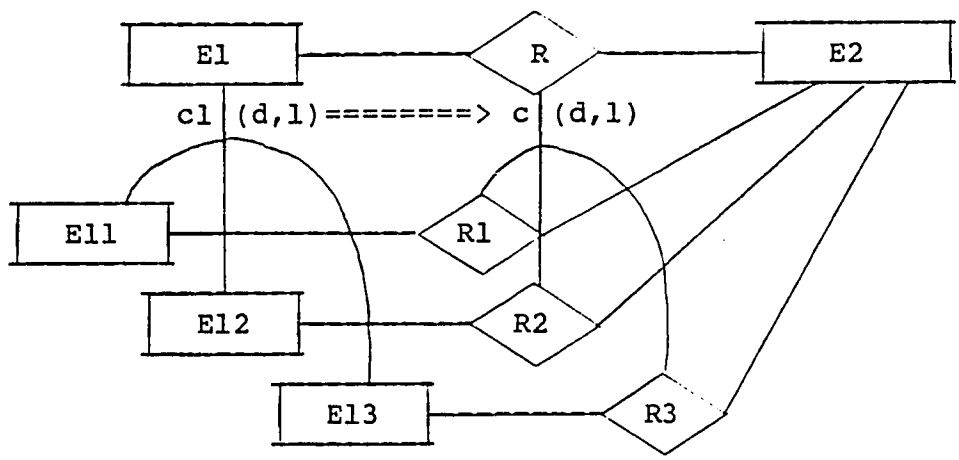
Furthermore, if there is one category ci under inducing E-set Ei (1<=i<=1) whose disjoint constraint indicates disjointness and each specialized R-set Rj relates to a unique successor of Ei then the disjoint constraint of c must indicates disjointness also. The following example shows that the R-splitting constraint of the category c under the R-set R indicates disjointness of successors. This is the direct result of disjointness of c2 and the fact that R1, R2 and R3 do not share common E-sets who are successors of E2.

Example 4.15



The mapping constraint of the constraint c of R may be determined by looking at the mapping constraints ci of Ei, i=1,...,1, which induce R. If the mapping constraints ci, i=1,...,1, are onto and all semantically feasible successors of R are selected then the mapping constraint of c is onto. The following example illustrates this point.

Example 4.16



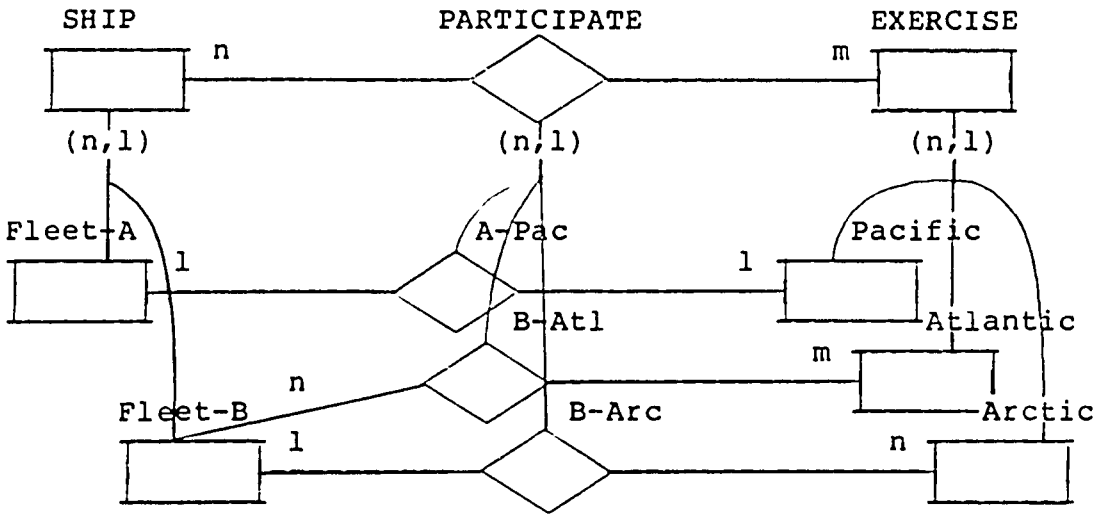
The combination of all structural constraints men-

tioned above give a database designer very powerful tools with which to work. Intricate and subtle semantic information of a real world system can be modeled into a database by these tools. The following example illustrates this point.

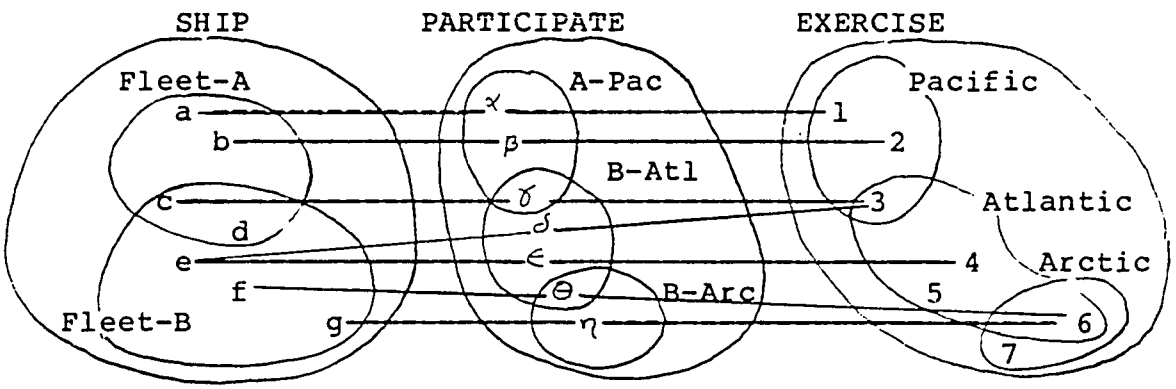
Example 4.17

The database in this example is the model of a small navy. This navy operates several ships and performs several peace time naval exercises. The ships and naval exercises are basic entities in the database. The ship entities are classified into the entity set SHIP and naval exercises into the entity set EXERCISE. These two sets are involved in a relationship set PARTICIPATE which contains information about ships participating in exercises. Additional semantic information is also included as various structural constraints. The database is represented by the two diagrams below.

Schema diagram



Instance diagram



Additional semantic information derivable from the schema diagram are:

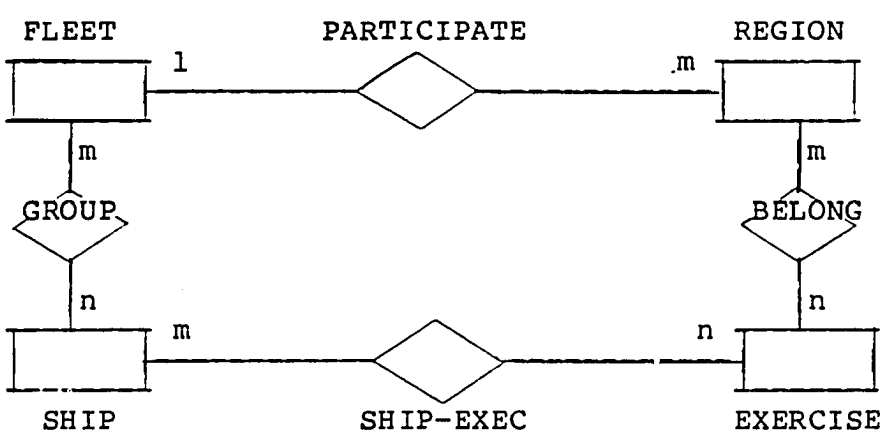
All ships are grouped into 2 non-disjoint classes Fleet-A and Fleet-B. And all naval exercises are grouped into 3 non-disjoint classes, Pacific, Atlantic, and Arctic. Any number of ships may participate in any number of naval exercises.

A ship in Fleet-A may participate in at most one naval exercise in the Pacific. And a Pacific exercise may have at most 1 ship from Fleet-A in it.

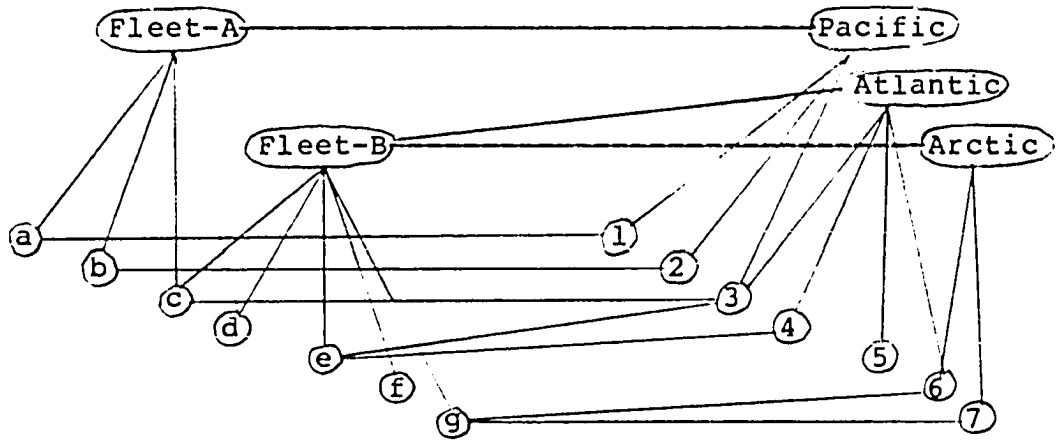
A ship in Fleet-B may participate in any number of exercises in Atlantic and Arctic areas. A naval exercise in The Atlantic may have any number of ships from Fleet-B in it. And a naval exercise in the Arctic may have at most 1 ship from Fleet-B in it.

The same navy database may be represented by the standard E-R model shown as the E-R diagram below. In this diagram, some information represented by the generic structures and structural constraints in the previous diagram is represented by different non-related links. We need several additional explicit constraints to enforce semantic integrity in the diagram below.

Schema diagram



Instance diagram



For example, to state that a naval exercise always belong to at least one region, the following explicit constraint must be included in the database.

if e is in EXERCISE then there exist [r,e] in BELONG where
r is in REGION.

To state that a ship in Fleet-B may participate in any number of exercise in Arctic region, the following explicit constraint is needed.

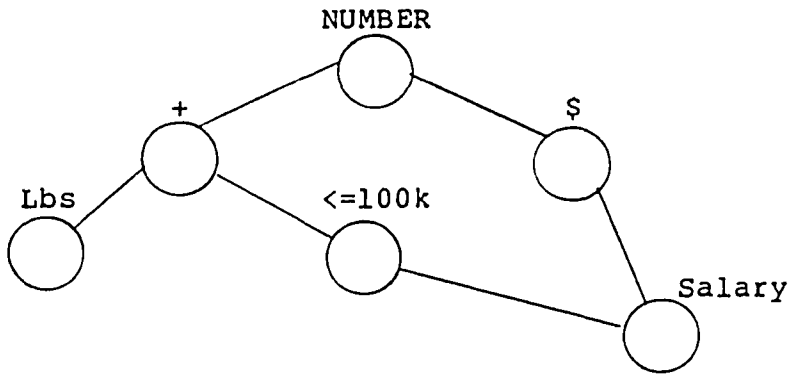
count(ei) >= 0, if [s,ei] is in SHIP-EXEC
and s is in SHIP and [Fleet-B,s] is in GROUP
and ei is in EXERCISE and [Arctic,ei] is in BELONG.

4.4 Schema and Operators of the Extended E-R Model.

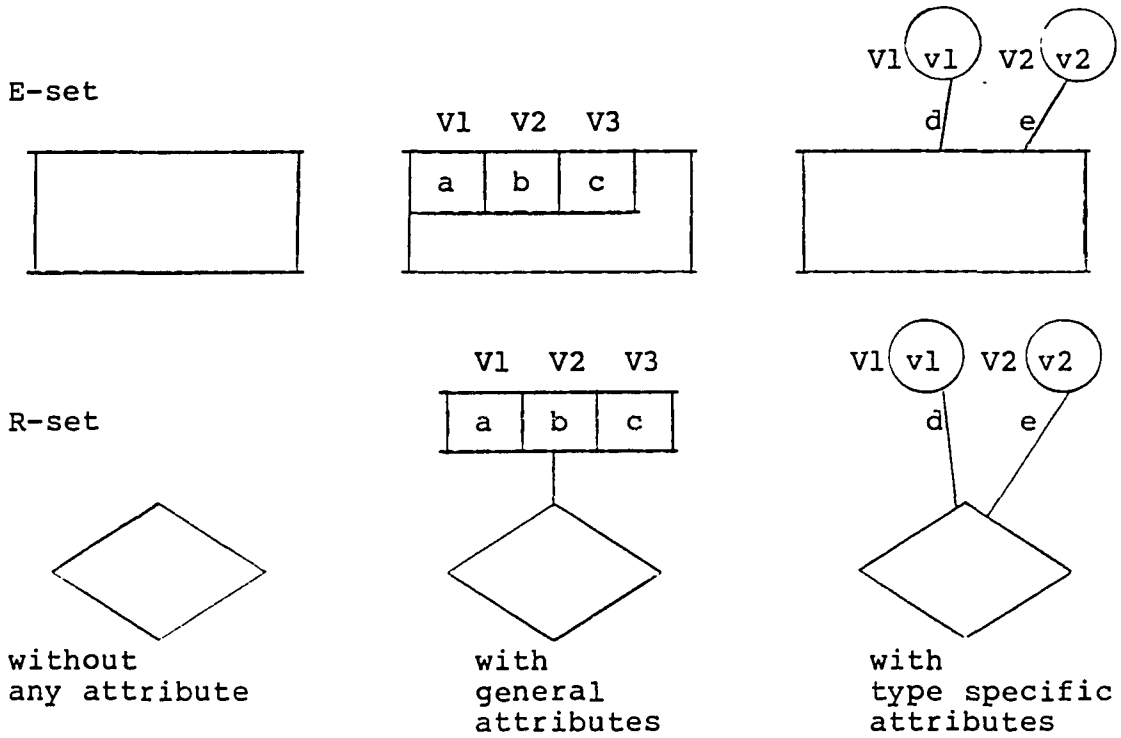
4.4.1 Extended E-R Diagram.

The original E-R diagram is modified to accommodate the change in the extended E-R model. Value sets and their generalization structure are represented by a V-set diagram separated from the rest of the schema. In a V-set diagram, a V-set is represented by a circle and a generic connection

is represented by an arc between two circles. An example of a V-set diagram is shown below.



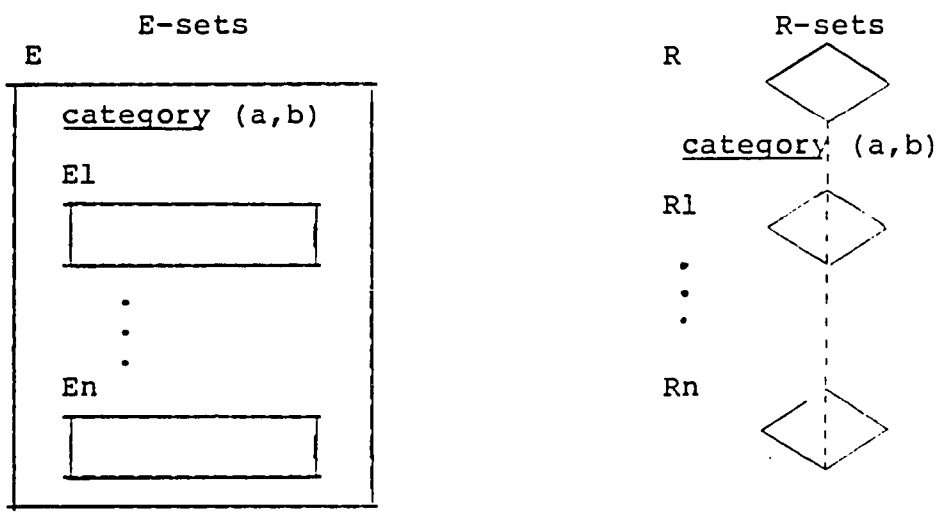
The remaining concepts of the E-sets, R-sets, generalization structures of E-sets and R-sets, attributes, V-sets and structural constraints are represented by another diagram called a schema diagram. In this diagram, the following symbols are used to represented these concepts.



Where
V1,V2,V3 are V-set names
a,b,c are general attribute names
d,e are type specific attribute names
and v1 and v2 are values.

The V-sets are not represented in the diagram as geometric symbols but are shown as strings. Generalization structures of E-sets and R-sets are shown in a diagram as follow:

Generalization of E-set and R-set:



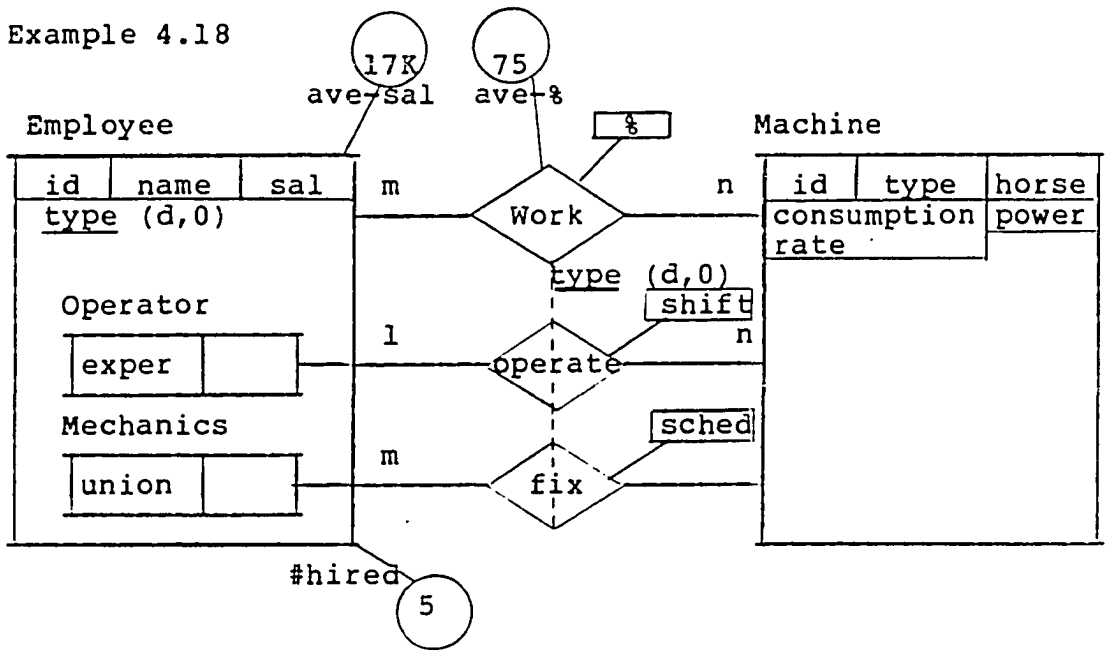
In each of the diagrams above, the ordered pair (a,b) written to the right of a category represented the splitting constraint associated with this particular generalization. The connectivity constraints associated with the R-sets are represented in an extended diagram the same way they are represented in a standard E-R diagram. There are certain items allowed to be omitted from an extended diagram and the information about them kept elsewhere. They are

all general attributes, all type specific attributes,

all value sets and all structural constraints.

The information about these missing items should be available to the users of the diagram when needed. The reason of allowing parts of a diagram to be omitted is to make it less complex and easier to understand. The following example is the schema of a small database using the extended diagram.

Example 4.18



4.4.2 Operators of the Extended E-R Model.

In this section we will examine retrieval operators only. Four primitive update operators: insert, delete, connect and disconnect are given in Appendix A. The operators for a database created in E-R model may be classified into two different classes, the set operators and the E-R algebra operators.

4.4.2.1 Set Operators.

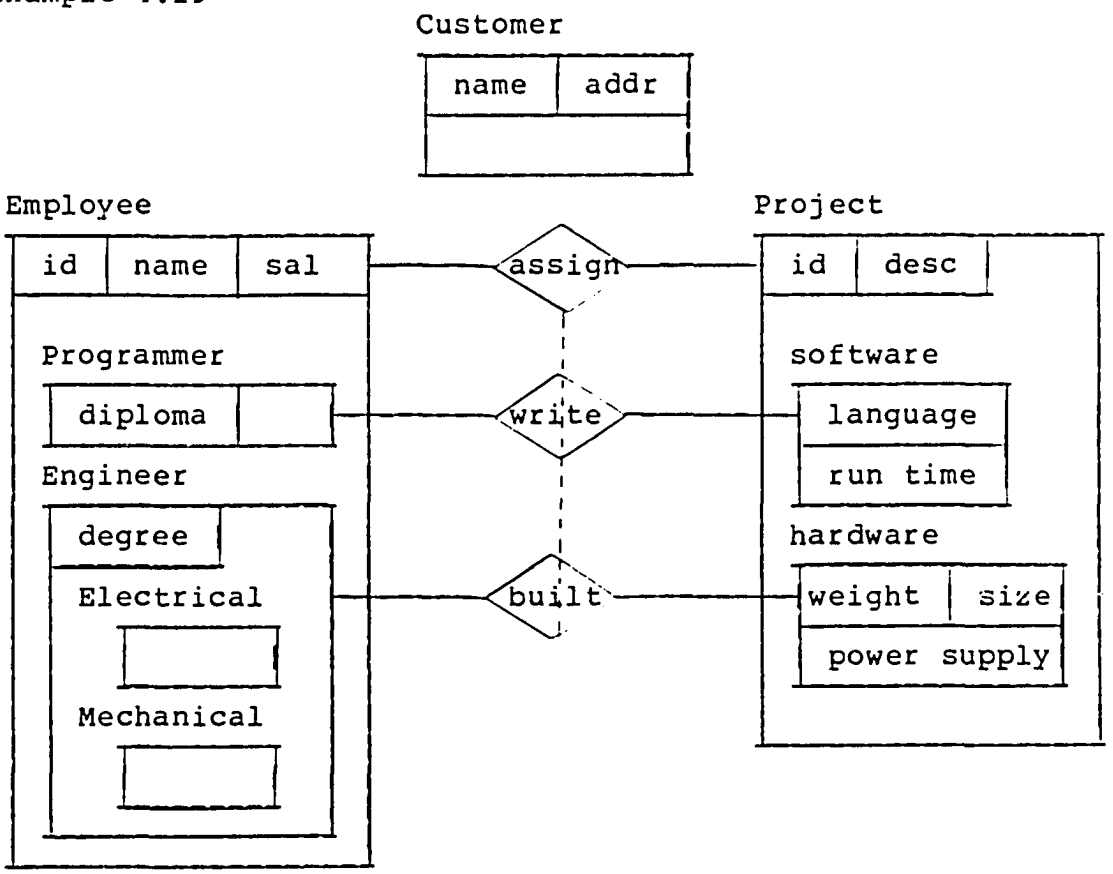
Because an E-R database consists essentially of families of sets, we allow the basic set operators, union, intersection, and difference to be used. For a relational database, these operators are defined on two union compatible relations. Two relations are union compatible when they have the equal number of attributes whose underlining domains are identical. We modify the concept of union compatible to be two different E-sets, Ei,Ej from FE or R-sets, Ri,Rj from FR that share at least one common predecessor along a generalization structure. In other words, they both belong to the same E-graph or R-graph and hence represent the same concept. The relationship between two "compatible" sets may be: one set is a sibling of another, or one set is an ancestor of another, or one set is an nth cousin of another. The result of a set operation is a new E-set Ek or a new R-set Rk, the new set inherits attributes and connections from one or several of the sets in the database. The following table shows attributes and sets inheritance rules.

operator	operand	inheritances
union	Ei,Ej	all general attributes of the closest common ancestor* of Ei and Ej
	Ri,Rj	all general attributes and all E-sets of the closest common ancestor* of Ri and Rj
intersection	Ei,Ej	all general attributes of Ei and Ej
	Ri,Rj	all general attributes of Ri and Rj and all E-sets of the closest common ancestor* of Ri and Rj
difference	Ei,Ej	all general attributes of the left operand Ei
	Ri,Rj	all general attributes and all E-sets of the left operand Ri

*-if two operand sets are related directly, i.e., one is an ancestor of another then the closest common ancestor is one of the operand itself. .ds

The following database will be used to show the results of different operators defined in this chapter.

Example 4.19



From this database, the following operations are defined

1) E1 <----- Electrical ∩ Mechanical
   operands are siblings and the result inherits all attributes from both operands,

2) E2 <----- Programmer ∪ Electrical
   operands are second cousins and the result gets all general attributes from the set Employee.

3) R1 <----- built - write
   operands are siblings and the result R-set gets all general attributes and related E-sets from built.

4) R2 <----- built ∪ write
   the same sibling operands, but this time R2 inherits all general attributes from both built and write and inherits related E-sets from assign.

4.4.2.2) E-R Algebra Operators.

We also define additional operators for the extended model following the tradition of relational algebra. This class of operators consists of select, project and join operators defined as follow.

Project - a projection is defined on an E-set Ei or an R-set Ri and a set of attributes al,a2,..,an, as

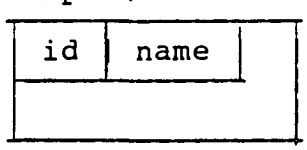
Ej <----- $\Pi_{al,a2,...,an}$ Ei or

Rj <----- $\Pi_{al,a2,...,an}$ Ri.

The result of this operation is a new E-set Ej or R-set Rj that has the same members as the operand set but all attributes except al,a2,...,an are disconnected from this set. Example of project is

Example 4.20

Emp <------- $\Pi_{id,name}$ Employee.



Join - a join is defined on two independent E-sets Ei and Ej as follows

Ek <----- Ei $\bowtie$ Ej. The resulting E-set Ek has all common elements of the sets Ei and Ej, and inherits all general attributes from both sets. The attributes with the same name are made unique by appending the set name in front. Example is

Example 4.21

Customer-Employee <----- Customer $\bowtie$ Employee.

id	employee.name	sal
customer.name	addr	

Select- a selection operator is defined on an E-set Ej or an R-set Rj and a set of attribute conditions as follows

Ek <----- $\sigma_{c1,c2,...,cn}$ Ej or

Rk <----- $\sigma_{c1,c2,...,cn}$ Rj.

Each ci (i=1,2,...,n) is a condition of the form
ai Θi vi, where
ai is an attribute name of the operand set,
Θi is a relational operator (=,=,<,>,<=,>=), and
vi is a value from the V-set Vi associated with attribute ai.

The resulting E-set, Ek or R-set, Rk has all element of the operand set that satisfy all conditions, c1,c2,...,cn.

Ek = { e | e is in Ej and ai:e--->v and
v Θi vi is true for i=1,2,...,n }.

Selection result of an R-set is defined in a similar fashion. An example of selection of an E-set is

Example 4.22

Rich-Employee <----- $\sigma_{sal > 50K}$ Employee.

id	name	sal

Selection via link is defined on an R-set R together with all of its relating E-sets E1,E2,...,En, as

T <------ SσR,E1,E2,...,En
where R,E1,E2,...,En are operand sets,
S is a selector which names one of the operand set,
and T is the target set (the result).

The result T is defined as

an E-set, if S = Ei (1<=i<=n)
T = { ei | e1 in E1 and e2 in E2, and ...
ei in Ei, and ... en in En
and [e1,e2,...,en] in R },

or an R-set, if S = R,

T = {[e1,...,en] | e1 in E1 and ... en in En
and [e1,e2,...,en] in R }.

This resulting set is a subset of the operand set specified by the selector and inherits all general attributes (and related E-sets if S=R) from that set. An example for this operation is the following

Example 4.23

Rich-Project <----- Projectσassign,Rich-Employee,Project.

id	name	desc

More complicated queries may be expressed by combining several operators. The E-sets and R-sets which are the results of some previous operations may be used as operands in subsequent operations to obtain further results. The following example shows operations required to find all the languages used by "John Doe" on all of his project.

Example 4.24

1) John-Doe <----- $\sigma_{\text{name="John Doe"}}$ Employee

2) John-Doe-Proj <----- Projectσassign,John-Doe,Project

3) John-Doe-Software <----- Software $\cap$ John-Doe-Proj

4) Answer <----- π_{language} John-Doe-Software.

4.5 Extended E-R Diagram Presentation Technique.

An extended E-R diagram can be used as a conceptual schema for an E-E-R database. In this case, a diagram represents a logical view of the database. There are, however, some inherent problems associated with using a diagram as a conceptual schema. A diagram, representing a large and complex real world system, is usually complex and difficult to use. A user must deal with large amount of information when he is presented with a diagram in its entirety. The remedy for this problem is to apply abstraction to the extended diagram. A user is presented with only the most general aspect of a database. Only the skeleton of a database with all details left out will be visible. In most cases, the initial view will consist of a few sets and connections and will be easier to comprehend. This first view is termed "the top level view" and is actually an abstract object of the entire diagram whose specific details are suppressed. The abstraction concept works rather well with the extended E-R diagram because the database itself is organized into multi-layer abstract objects according to the generalization structures of the E-sets and R-sets. The top level view is supplemented by several "diagram operators" which provide the user with the tools to selectively reveal additional de-

tails of the part of a database he is interested in.

4.5.1 How to Create the Top Level View.

After an extended diagram is created to model a particular system, the following procedure may be used to derive the top level view from the diagram.

1) Choose the most general E-sets to be included in the top level view. A most general E-set Ej is an E-set without predecessor, i.e., there does not exist Ei such that (c,Ei,Ej) is in GE.

2) All R-sets relating exclusively to the most general E-sets are also included in the top level view.

3) Mark each E-set Ej in the top level view by a shaded triangle in the top right corner if Ej has any successors, i.e., there is some Ek such that (c,Ej,Ek) is in GE.

4) No other E-sets and R-sets are included in the top level view.

5) No attributes, value sets and structural constraints are included in the top level view.

4.5.2 Diagram Operators.

This set of special operators may be used to apply to a current view. A current view is initially the top level view but later on is the result of applying one of the operators to some previous current view.

Show-attr(S) - shows all the attribute names associated with the set S, S may be an E-set or an R-set in the current view.

Omit-attr(S) - suppresses all attribute names associated with the set S, from the current view.

Show-value(S,a) - shows the V-set name of the attribute a of the set S; the operand set may be an E-set or an R-set in the current view.

Omit-value(S,a) - omits the V-sets name of the attribute a of the set S from the current view.

Show-cons(S,c) - shows the splitting constraints of the set S (an E-set or an R-set) under category c.

Show-cons(R) - shows the connectivity constraints associated with the R-set R.

Omit-cons(S) - omit all constraints associated with the set S (an E-set or an R-set) from the current view.

Set-level(n) - set up the level of abstraction allow to appear in one view, n is >= 2.

Zoom-in(E) - displays all specialized E-sets of the operand set in the current view. (specialization of an E-set)

Zoom-out(E) - displayed all generic E-sets of the operand set in the current view. (generalization of an E-set)

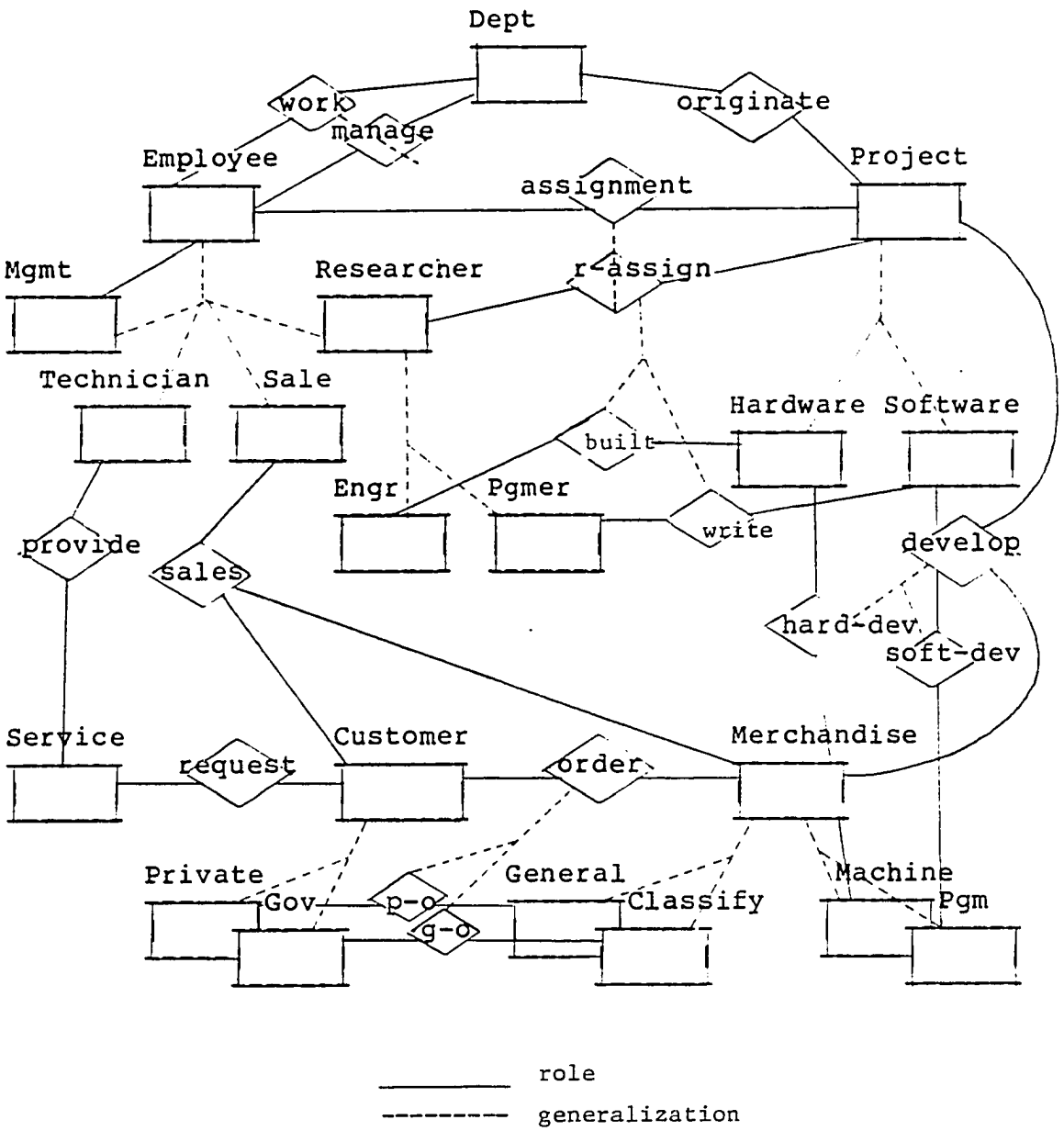
After applying the operators Zoom-in or Zoom-out, some E-sets may be added to the current view and some may be removed. If all the E-sets in the resulting view include the E-sets participate in an R-set not already shown then that R-set is also included in the view. If after eliminating some E-sets from the current view as the result of Zoom

operation, some R-set no longer has all of its E-sets in view, then that R-set is also removed from the view.

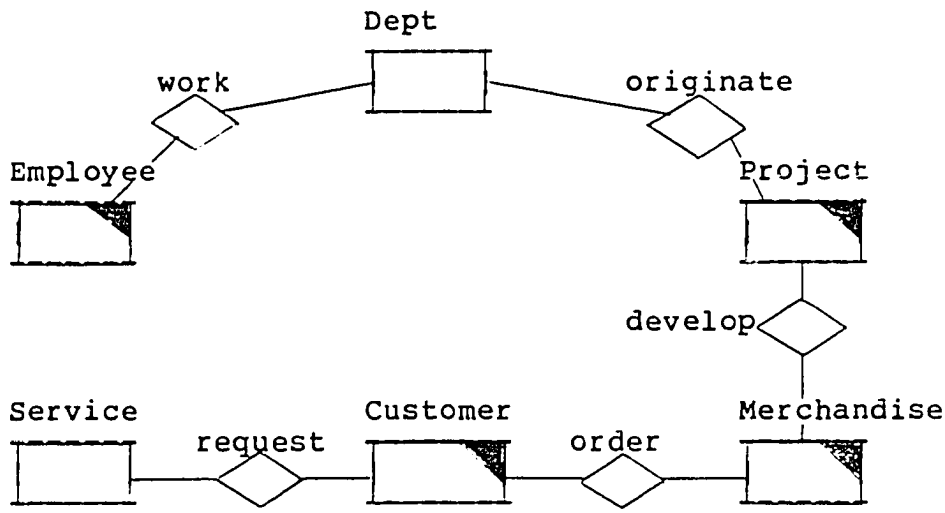
The nature of the diagram and its operators indicates that the ideal user's interface should be a graphic system. This type of access facility allows a diagram to be displayed, modified and operators to be applied conveniently. The following example shows the original diagram, the top level view and several subsequent views after some operators are applied.

Example 4.25

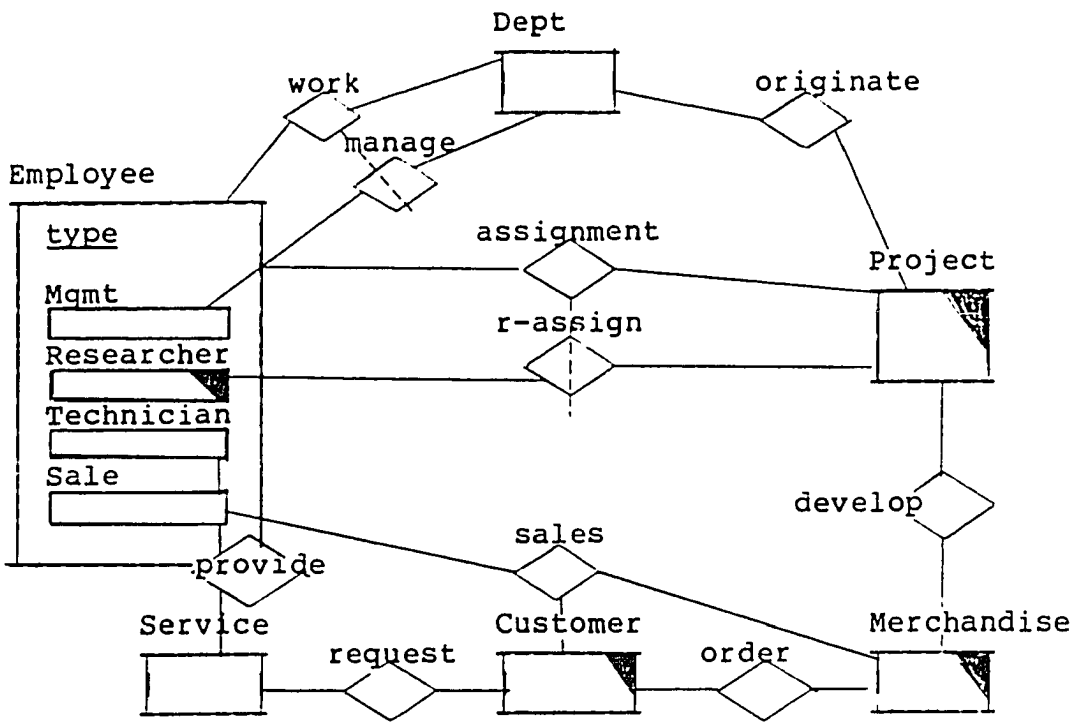
Original diagram - all attributes, V-sets and constraints are omitted for clarity.



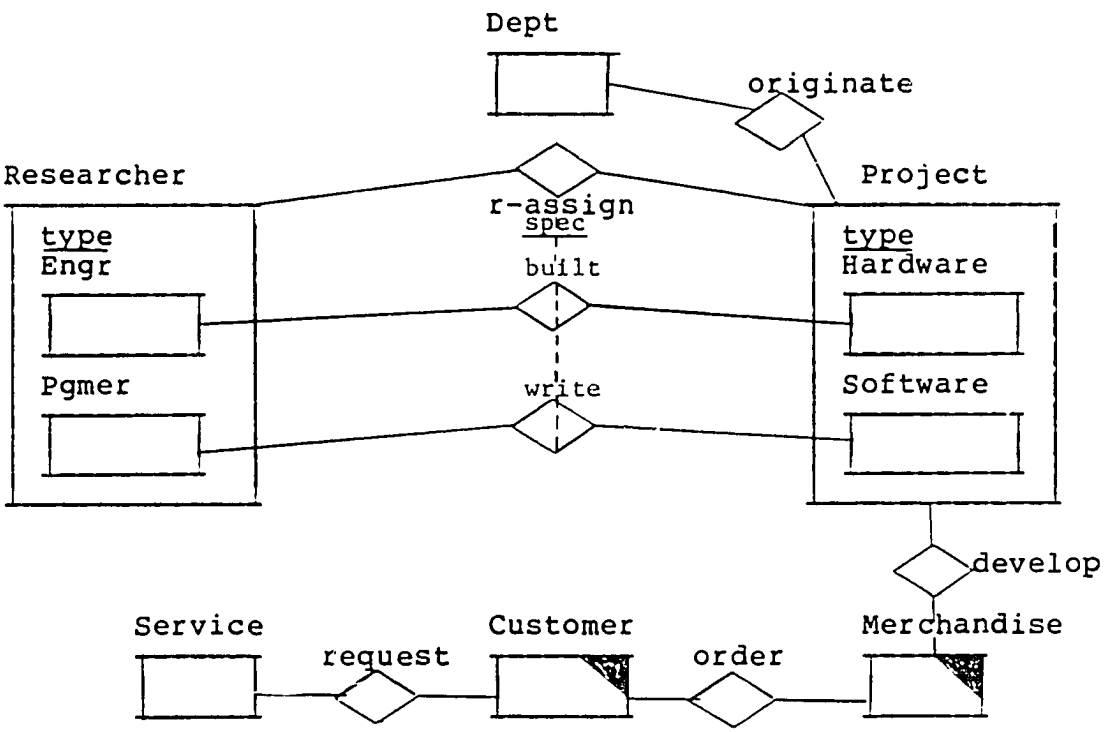
top level view (view 1)



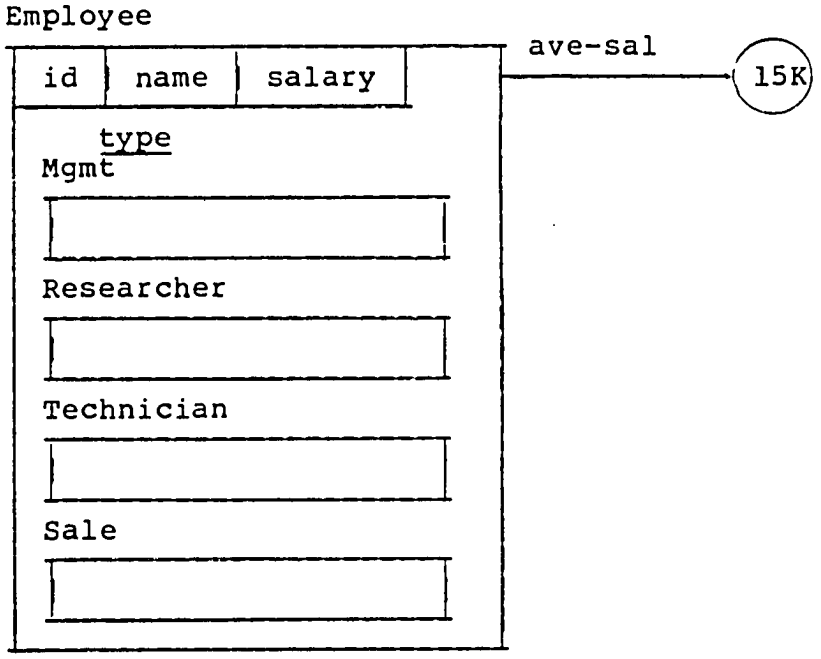
the view after applying Zoom-in(Employee) (view 2)



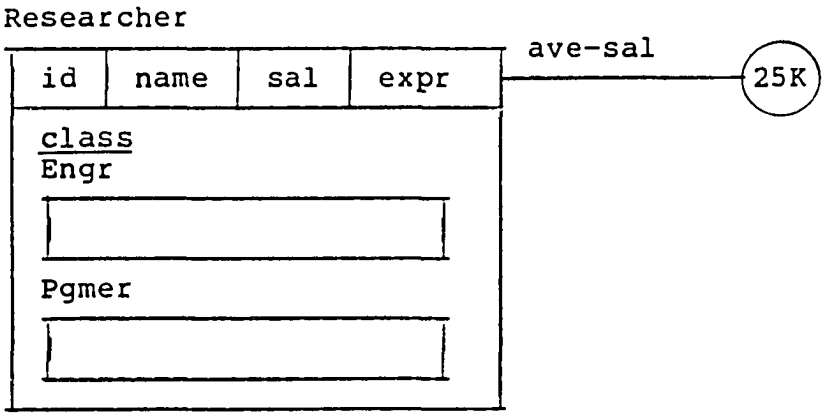
the view after applying Zoom-in(Researcher) and Zoom-in(Project) (view 3)



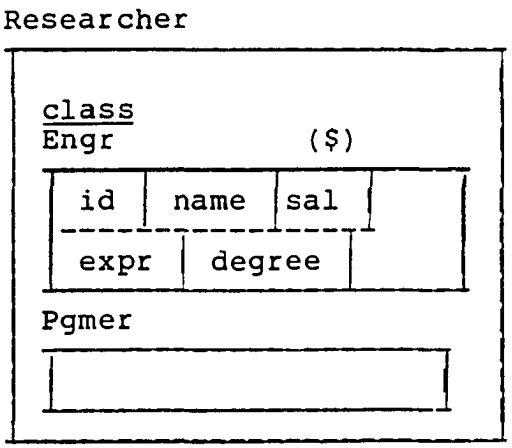
the view (restricted) after applying show-attr(Employee) to view 2



the view (restricted) after applying show-attr(Researcher) to view 3



the view (restricted) after applying show-attr(Engr) and Show-value(Engr,salary) to view 3



CHAPTER FIVE

DATABASE DESIGN TECHNIQUE

5.1 Introduction.

The primary objective of all database design is to organize the semantic information of the data in some orderly manner. In the extended E-R model, we try to capture related semantic information within one object (an E-set or an R-set). This process is similar to the normalization techniques of the relational model. In the relational model, relations are said to be in different normal forms. In one normal form, all dependencies of a certain type in a relation are implied by the keys of that relation. All relational normalization techniques obtain higher level normal form relations by decomposition. The primary objectives of any relational normalizations are reduction of data redundancy and elimination of update anomaly [C7]. However, in some normalization technique, the decomposition is carried out to the extreme. In some cases, the results are all binary relations. These techniques tend to produce a large number of very small relations which are not suitable for our model. In the extended E-R model, if E-sets and R-sets are split(decomposed) into several small sets, these sets will form interconnected structures. If these structures are complex, then they are expensive to use, e.g., an appli-

cation may required accesses to multiple sets in a database which may not be necessary if we keep a large set intact instead of split it up. So, in addition to consider underlying semantic information, a designer should try to keep the overall structure simple by maintaining a small number of sets in a database. A simple structured database is easy to understand and inexpensive to use and maintain. These two requirements should be balanced by the database designers. On one hand, we have to decompose E-sets and R-sets to organize semantic information. On the other hand, overdecomposition will fragment the database and create complex structures which are undesirable in our model.

5.2 Designing Process.

5.2.1 Objects Identification.

The designing process of a database in the extended E-R model should be done in a top down fashion following the designing process of the standard E-R model [C6]. The designer should collect the required information across all the user groups. From the information collected, the designer identifies the E-sets in the database and their properties (attributes and V-sets). Each E-set should be identified based on the following rules.

1) An E-set should be a stable concept which doesn't change as application requirements change.

2) An E-set should be independent of other concepts, i.e., the existence of an entity in an E-set should not

depend on the existence of other entities in any E-sets.

3) A special type of E-set, called weak entity set, is the exception to the second rule. The existence of an entity in a weak E-set is dependent on the existence of at least one entity in another E-set called strong E-set. The dependency is explicitly shown as a connection in a weak R-set.

After all the E-sets are identified, the designer should identifies the R-sets in the database and their properties (attributes and their V-sets). Each R-set is designed based on the following rules.

1) An R-set is a transient concept comparing to an E-set, so an R-set may be created to satisfy some application requirements.

2) An R-set is always dependent on the E-sets it related; therefore, a relationship in an R-set may exist only if all participating entities exist.

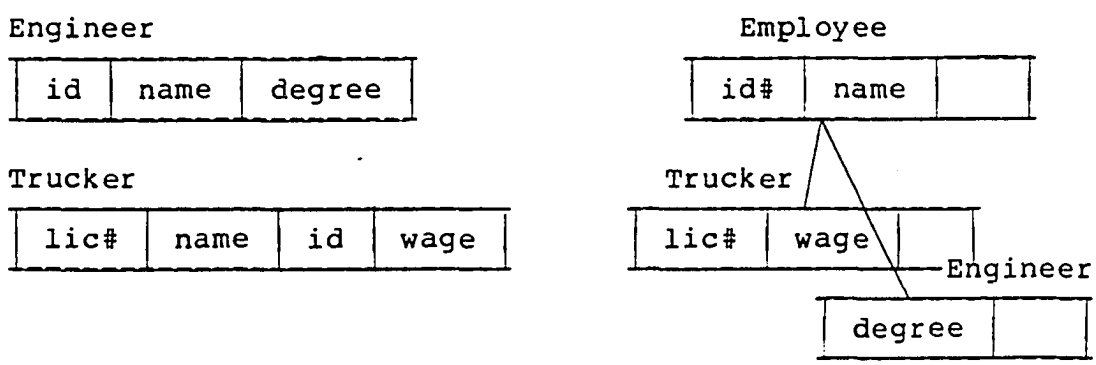
5.2.2 E-graphs and R-graphs Creation.

After all E-sets, R-sets and their properties in the database are identified, the designer may create different E-graphs and R-graphs from these sets. A graph structure may be created from a single set or a collection of sets. The graph is then refined or normalized based on inapplicability of attributes and data dependencies that exist in each node of the graph. The normalized graph has a structure that better represents the underlying semantic information. Hence, it is more suitable to be used as part of the

database schema in the extended E-R model.

E-graph Creation - An E-graph is created from the root node. The root node may be an E-set selected from the collection of all E-sets identified or a new generic E-set created from several E-sets. If a new root node is created, all E-sets selected must have some compatible attributes and must be considered by the designer to represent the same concept. Two attributes of two E-sets are considered compatible if they are functions to the same V-set or the same set of V-sets. After the new root is created, the E-sets representing the same concept are included in the graph as immediate successors of the root and all compatible attributes are moved up to the root.

Example 5.1



After the root of an E-graph is created, apply the following procedure to each node in the E-graph starting from the root node.

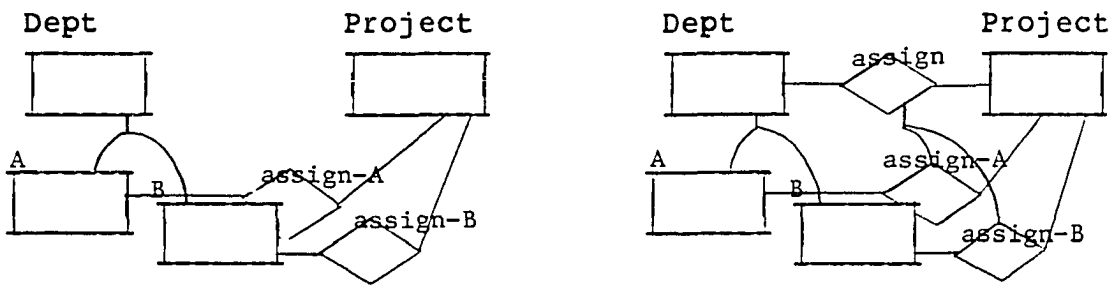
1) Apply categorization process (5.3) to the node in the graph, this process may extend the graph by creating a

set of immediate successors to the current node.

2) Identify context dependent dependencies (FDs) within the node. Then apply normalization process (5.5) to the node.

R-graph Creation - From the collection of all R-sets, create the root node of an R-graph. The root node may be an R-set selected from the collection of all R-sets previously identified or a new generic R-sets created to represent several R-sets. These R-sets must represent the same concept as determined by the designer. They must also have some compatible attributes and compatible composite keys. Composite keys of two different R-sets are compatible if the number of E-sets in each key are equal, and the corresponding pair of E-sets drawn from both keys must be the same or sibling E-sets. The new root node and all R-sets representing the same concept form an R-graph similar to the way the E-sets representing the same concept form an E-graph.

Example 5.2



Apply the following procedure to each R-node in the R-graph starting from the root node.

1) Apply categorization process (5.3) to the node, a

new set of successor nodes may be created as a result of this process. In some cases, additional E-nodes must be created in the surrounding E-graphs to satisfy the inducing E-set rule of an R-set.

2) Identify context dependent dependencies (FDs and MVDs) within the R-node. Then apply normalization process (5.5) to the node.

5.2.3 Splitting Constraint Assignment and Graph Analysis Process.

After obtaining the E-graphs and R-graphs, assign the E-splitting constraints and R-splitting constraints to them based on splitting constraint assignment process (5.4). Then apply graph analysis process (5.6) to each E-graph and R-graph. If the graph analysis process detects some conflicting structural constraints of a graph, then the designer must modify these constraints. Finally, assign the connectivity constraint to each node in every R-graph. The assignment should be based on the mapping characteristics of the R-set represented by the node (see APPENDIX B).

5.3 Categorization Process.

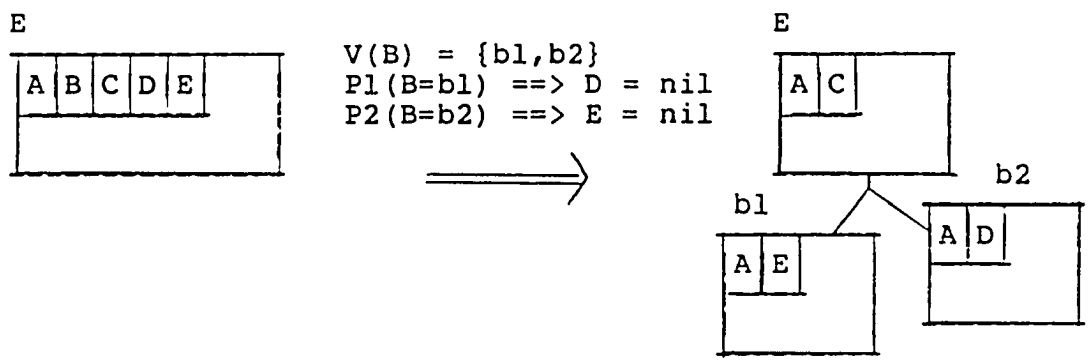
Categorization of an E-node in an E-graph or an R-node in an R-graph should be based on the following guidelines.

5.3.1 E-node Categorization.

1) Do content analysis based on inapplicability of attributes. Let E(A1,...,An) be an E-node and P1,P2,...,Pm be

a set of predicates which determine subsets E1,E2,...,Em of E. If each Ei (i=1,...,m) has certain attributes which are always null (inapplicable), then E may be specialized to E1,...,Em. Each node of E1,...,Em has only the attributes of E which are applicable to it.

Example 5.3

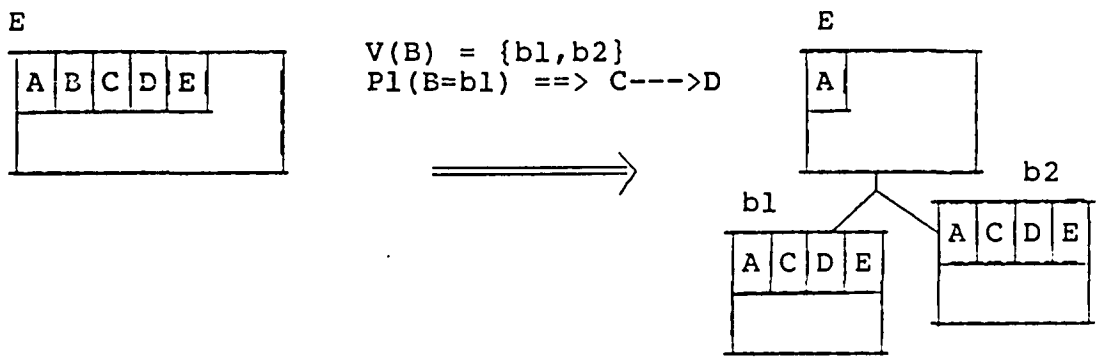


Example 5.3 shows that if all predicates are simply conditions on the same attribute and each predicate is related to a unique value of the attribute, then this attribute may be removed because the same information may be represented by a generic connection. In this case, each value may be used as a specialized E-set name.

2) An embedded FD in a subset of an E-node may be eliminated by specialization. Let E(A1,...,An) be an E-node and P be a predicate which determine a subset E1 of the node E. If an FD Ai--->B such that AiB $\subseteq$A1,A2,...,An holds in E1 but not in E, then the FD is an embedded FD. If Ai--->B is non-trivial and is not implied by P, then it will causes the same type of data redundancy and update anomalies as seen in relational database [S4]. To eliminate this embedded FD

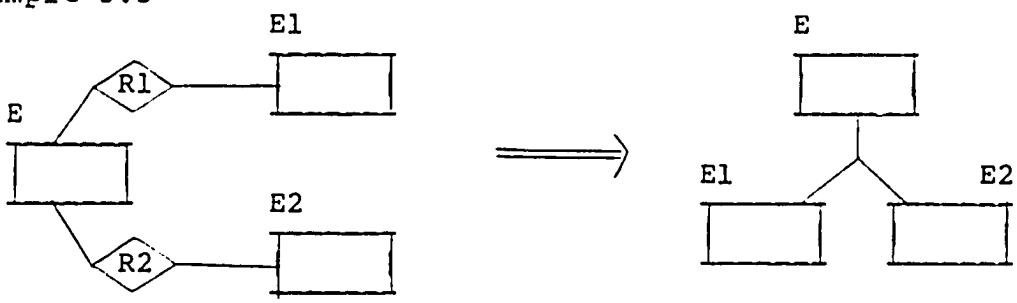
from the E-node, we add two new node E1 and E2 as immediate successors of E where E2 is E - E1. As a result, the set E(A1,...,An) is replaced by the sets E(A1,...An - AiB), E1(A1,...,An) and E2(A1,...,An). The FD Ai--->B now exists only in E1 and may be eliminated by some other mean.

Example 5.4



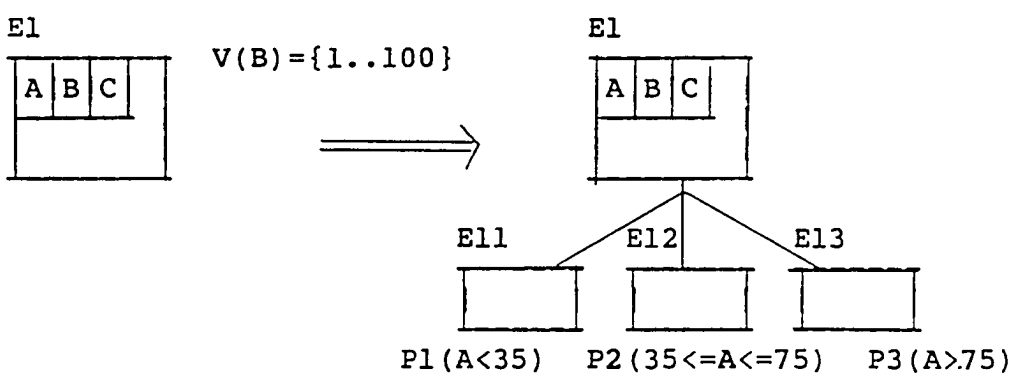
3) Converting several one-to-one relationships among several E-graphs into generic connection of one E-graph. Let E,E1,E2,...,En be root nodes of different E-graphs and R1,...,Rn be one-to-one binary R-sets between E and E1, E and E2, ..., E and En respectively. If E1,E2,...,En represent different aspects of the concept represented by E, then we may convert E1,...,En into immediate successors of E. If Ri has any attributes, then they must be moved to Ei.

Example 5.5



4) A specialization of an E-node may be done to satisfies an application requirements. A specialization of this type is usually simple grouping based on attribute value or grouping induced by different R-sets.

Example 5.6



5.3.2 R-node Categorization.

1) An R-node may be categorized based on inapplicability of its general attribute or the presense of an embedded FD among its general attributes the same way that an E-node categorization is done. With the exception that, when a specialized R-node is added to an R-graph, we may have to create additional E-nodes in the surrounding E-graphs. These specialized E-nodes are needed to satisfy inducing E-

sets rule of the specialized R-set.

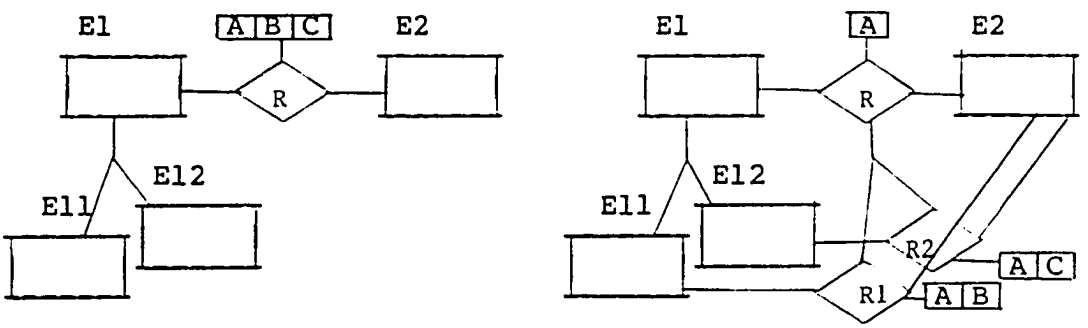
Example 5.7

Assuming that the following conditions hold in this example database shown in the left diagram:

1) P1([e1,e2] in R and e1 in E11) ===> C=nil

2) P2([e1,e2] in R and e1 in E12) ===> B=nil.

Then we may categorized R into R1 and R2 as shown in the right diagram.



2) An R-node could be categorized according to some application requirements. The specialization of this type may be simple grouping based on attribute values or grouping induced by some surrounding E-nodes. As in the previous case, additional E-nodes may have to be created to satisfy inducing E-sets rule.

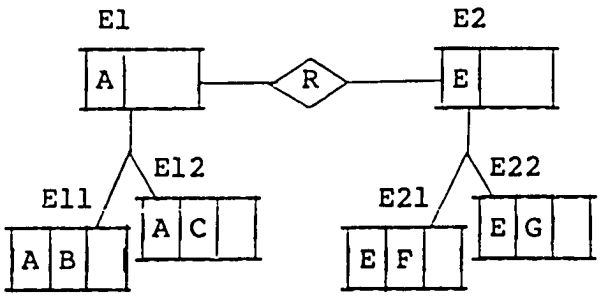
Example 5.8

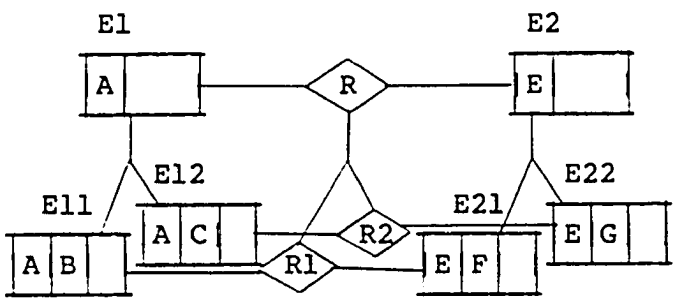
Assuming the following application requirements for the database represented by the top diagram:

1) the attributes A,B and F are accessed together through R, and

2) the attributes A,C and G are accessed together through R.

Then the R-set R may be specialized into R1 and R2 as shown in the bottom diagram.





The categorization process for an E-node or an R-node may be done as a combination of different rules. The primary objectives of the categorization process are the following:

1) To eliminate an embedded FDs from a subset of a node.

2) To eliminate null value from inapplicable attributes of a node.

3) To facilitate application requirements by putting together attributes which are frequently accessed together.

4) To simplify the top level view of a schema by

creating a hierarchical type structures.

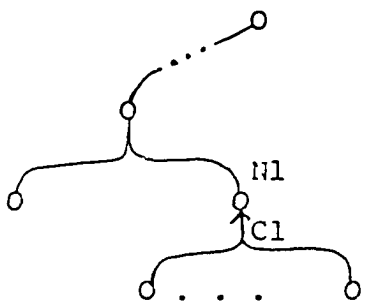
5) To isolated different concepts into different graph structures which facilitate localization of changes.

5.3.3 Different Type of Categorizations.

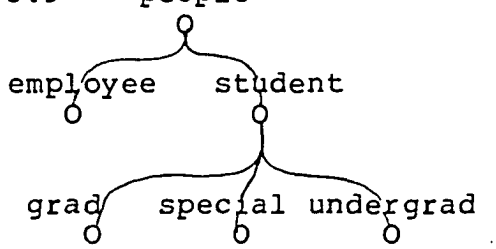
There are five different types of categorization which can be done to a node in an E-graph or an R-graph.

Type I. Independent Categorization

We say that the category C1 of the node N1 is an independent categorization if a set of new nodes are created as immediate successors of N1 under C1 only.

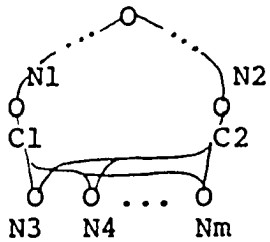


Example 5.9

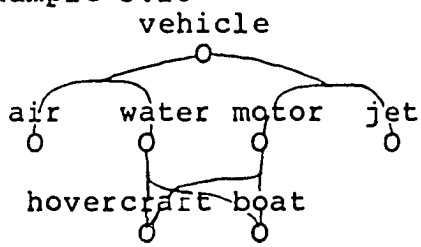


Type II. Fully Dependent Categorization

We say that the category C1 of the node N1 is fully dependent on the category C2 of node N2 if N1 and N2 have all immediate successors in common. Fully dependent categorization is only allowed between non-sibling nodes or between non-disjoint sibling nodes.

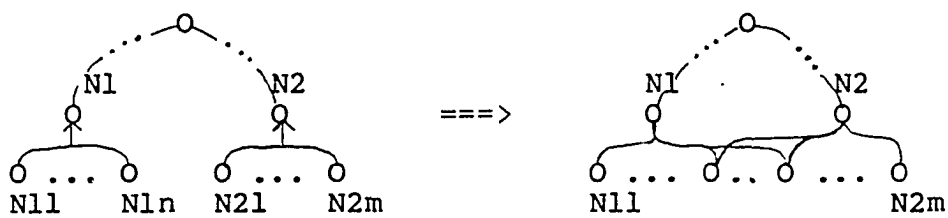


Example 5.10

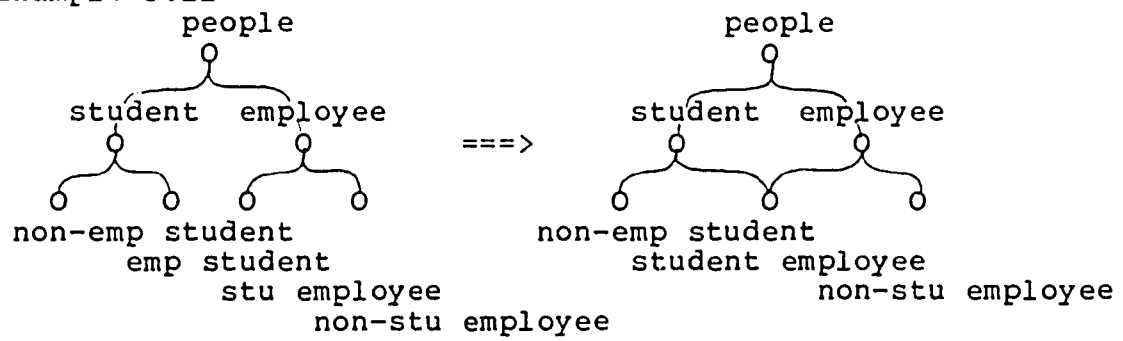


Type III. Partially Dependent Categorization

We say that the category C1 of the node N1 is partially dependent on the category C2 of the node N2 if N1 and N2 share some immediate successors. Let the sets of siblings S1={N11,N12,...,N1n} and S2={N21,N22,...,N2m} be immediate successors of N1 under C1 and N2 under C2 respectively. If some members of S1 and S2 are the same sets, then they may be merged together. These merged sets have both N1 and N2 as immediate predecessors.



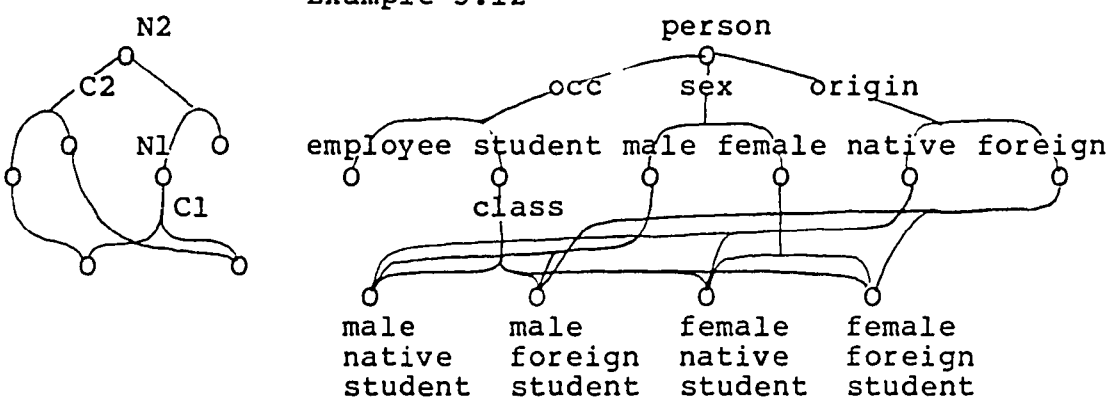
Example 5.11



Type IV. Fully Induced Categorization

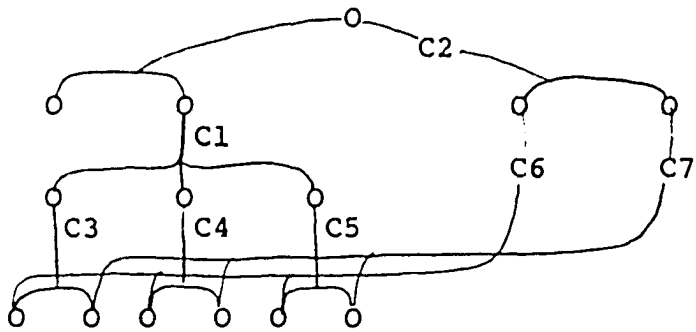
We say that the category C1 of the node N1 is induced by the category C2 of the node N2 if a set of new nodes are created as siblings under C1 of N1 are also the direct successors of all sibling nodes under C2 of N2.

Example 5.12



class of student is induced by sex and origin of person

We also allow mutually induced categorization between two or more categories. In the diagram below, C1 and C2 mutually induce each other.

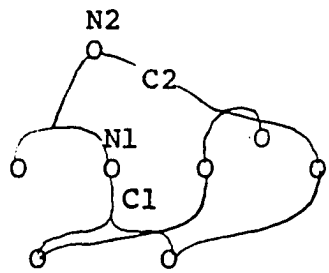


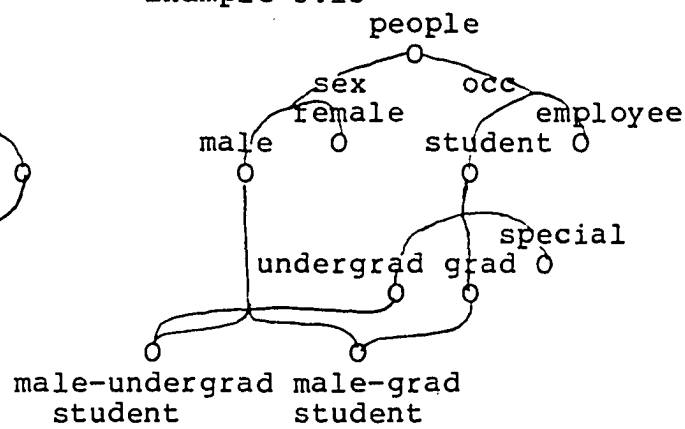
C3, C4, C5 are induced by C2, and
C6, C7 are induced by C1

Type V. Partially Induced Categorization

The partially induced categorization is defined to be the same as fully induced categorization except that not all of the siblings under the category C2 of the node N2 are required to be the direct predecessors of the new node created under C1 of N1.

Example 5.13





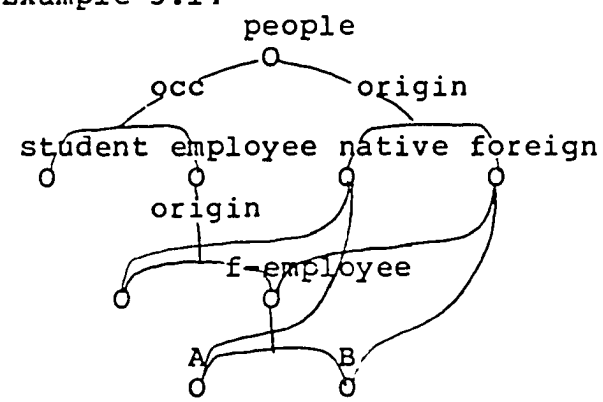
C1 of N1 is partially induced by C2 of N2

student-type of male is partially induced by type of student

Inducement Rule.

A node which is created as a direct or indirect result of inducement by the category C of the node N should not be induced again by C. If this type of categorization is done the graph will contain a null subgraph under the node in question.

Example 5.14



f-employee is created as a direct result of inducement by origin of people, applying the same category to induce f-employee causes the left successor of f-employee to be null.

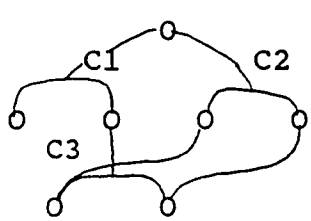
5.4 Splitting Constraint Consideration.

After the entire E-graph or R-graph is designed, the splitting constraints of each category in the graph should be assigned. The constraints assignment should be done in the top down style. Starting from all of the categories under the root node, proceed downward in breadth first manner. The mapping constraint of each category should be specified first. The specification is based on the type of mapping between a node and its immediate successors. The disjoint constraint is then specified based on the relation-

ship among sibling nodes under the category. An E-splitting constraint assignment is based on the underlying semantic of the particular generalization connection. However, an R-splitting constraint assignment is derived from the E-splitting constraints of the inducing E-nodes in the surrounding E-graphs. This derived constraint may be modified by the designer in some cases (see appendix-a).

The disjoint constraint of a certain type of category is influenced by the disjoint constraints of other categories in the same graph. Only the disjoint constraint of an independent category or a partially dependent category may be specified independently. The disjoint constraints of all categories which are fully dependent on each other must be the same. And finally, if one category is induced (fully or partially) by other categories all of which are disjoint, then the induced category must be disjoint also. If at least one of the inducing category is non-disjoint then the result may be disjoint or non-disjoint depending on other factors. A non-disjoint induced category may have additional restriction on the possible instantiation due to constraints of the inducing categories.

Example 5.15

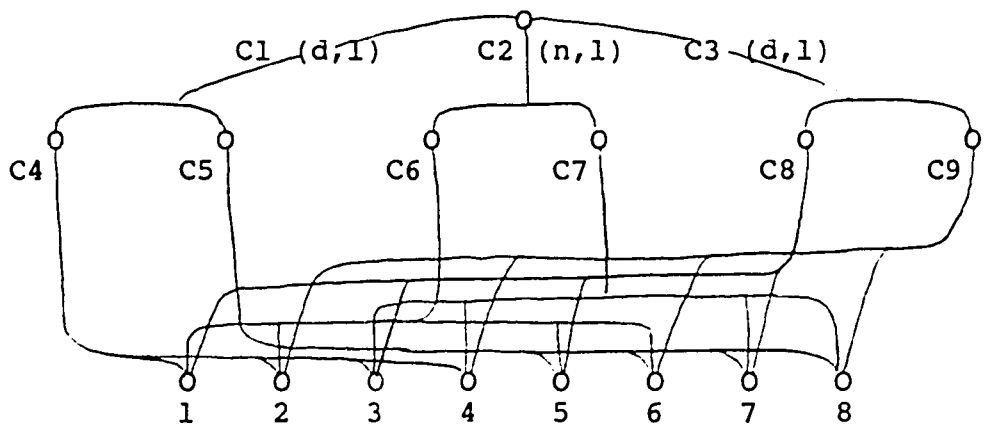


If C2 is disjoint then
C3 must be disjoint also.

If C2 is non-disjoint then C3 may
or may not be disjoint.

In the case of mutually inducing categories, the same rule is still applicable to each of the resulting categories. An example of mutually inducing categories is shown in example 5.16.

Example 5.16



C6 is induced by C1 and C3 ==> C6 is disjoint
C4 is induced by C2 and C3 ==> C4 may be disjoint or non-disjoint.
If C4 is non-disjoint only one element of the set {1,2,4,5,(1,4),(2,5)} may be instantiated.

5.5 Normalization Process.

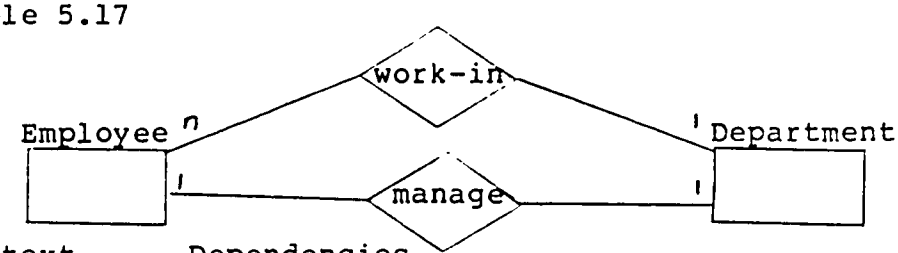
Normalization of an E-set or R-set is based on the context dependent dependencies of the set.

5.5.1 Context Dependent Dependencies.

We will consider three types of data dependency: the functional dependency (FD), the multi-valued dependency (MVD), and the embedded multi-valued dependency (EMVD). The definitions of these types of data dependencies can be found in [A1, B2, B4, C7, D1,K3, U1]. In the extended E-R model,

a data dependency exists within a context. A context is either an E-set or an R-set with its surrounding E-sets. In a context of an E-set, we consider the FDs that exist among the attributes. In a context of an R-set, we consider the FDs among the attributes and the MVD or EMVD that may exist among the key attributes representing relating E-sets. The same set of attributes may form several contexts each of which has a unique set of dependencies as shown in the example below. This assumption is different from the universal relation assumption in the relational model.

Example 5.17



Context Dependencies
work-in: {employee-->department}
manage: {employee-->department; department-->employee}

Decomposition and specialization of a set as a result of normalization will preserve all dependencies of the original context. Decomposition of an embedded object from a set will preserve dependencies by means of a new R-set and, sometimes, additional explicit constraints. Specialization of an E-set or an R-set by categorization process also preserve dependencies of the original context by mean of properties inheritance and generic connections between a node and its successors. The only time the original context is not preserved is when an R-set is determined to be compo-

sited due to an EMVD or an MVD. If a composite R-set is split, then the original context will be represented by two new contexts.

5.5.2 Decomposition of an Embedded Object form an E-set.

Given an E-set Ei(X), where X is the general attributes of Ei. And let Fi be a minimum cover of FDs among X (a minimum cover Fi is a minimum set of FDs among X from which all valid FDs of Ei among X can be derived [Ul]). An embedded object of Ei is a set of attributes YZ which is a subset of X, where $Z \not\subseteq Y$ and $YZ \not\rightarrow X$ (Y-->Z is not a trivial FD and YZ doesn't contain a key of Ei).

case I. YZ is taken directly form Fi, i.e.,

YZ is an embedded object if Y-->Z $\in$ Fi.

case II. YZ is taken from several members of Fi, i.e.,

YZ is an embedded object

if Y-->Z1, Z1-->Z2,..., Zm-1-->Zm $\in$ Fi

and Z = Z1$\cup$Z2 ...$\cup$Zm.

The second case represents multiple transitive dependencies where we have embedded objects within embedded object. The embedded object within an embedded object should be removed at a later stage to simplify the overall structure.

After an embedded object is found, a decomposition is done by projection similar to the relational decomposition. If YZ is an embedded object of Ei(X), then decompose Ei(X) into

Ej(X-YZ), Ek(YZ) and Rl(Ej,Ek:) where card(Rl) is (n,1).

5.5.2.1 Functional Dependencies Preserved by an A Decomposition.

After a decomposition of Ei the FDs of the original context are projected onto the sets, Ej and Ek. The projected FDs of Fi onto Ej and Ek are called Fj and Fk respectively, i.e.,

$$Fj = \Pi_{X-YZ} Fi, \text{ and}$$

$$Fk = \Pi_{YZ} Fi.$$

Since the connectivity constraint of Rl is specified as (n,1), the FD from the primary key of Ej to the primary key of Ek exists in the R-set, Rl. Let Kj and Kk be the primary keys of Ej and Ek respectively, all of the general attributes YZ in Ek are transitively dependent on Kj through Kk. The FD Kj-->Kk in Rl will be called Fl. If all of the FDs in Fi are implied by all of the FDs in Fj and Fk and Fl, then the original context dependent dependencies of Ei are preserved after the decomposition. Otherwise the process called F-maintenance is required to preserve the original FDs in the context Ei after the decomposition.

5.5.2.2 F-maintenance After Decomposition of an Embedded Object.

If after a decomposition of an embedded object, Fi is not preserved by the dependencies in Fj, Fk and Fl then F-maintenance process is required. This process is done by

1) adding explicit constraints, and/or

2) adding duplicate attributes in Ej and Ek.

Decomposition is done to each E-set until no more embedded object is found or the designer decide to terminate the process. To balance the designing objectives, we recommend that the decomposition be carried out for each embedded object whenever the F-maintenance process is not required. If F-maintenance is required, then the embedded object should be kept within the original set. And the dependency or dependencies that cause the embedded object should be enforced by other means. After decomposition, the embedded object may be merged into existing E-graph if they represent the same entity concept.

5.5.2.3 Lossless Join Property of a Decomposition.

In the relational model the lossless join property is extremely important. The lossless join property indicates that the original relation can be recovered from several relations which are the result of the decomposition of the original relation. In the extended E-R model, any decomposition of an embedded object always has a lossless join property. This claim originates from the fact that, when an embedded object is decomposed from an E-set, the resulting E-sets are not fully detached from one another. A new R-set is always created to tie the two E-sets together, and the original E-set is always recoverable through this R-set. If the decomposition of an embedded object from an E-set is observed in terms of the relational model, the original rela-

tion Ei(X) is decomposed into 3 relations Ej(X-YZ), Ek(YZ), and Rl(Kj,Kk:). Kj and Kk represent the primary key attributes of Ej and Ek respectively. We can create a tableaux for this decomposition as follow.

Kj	Kk	X-YZ	YZ
al	b12	a3 ...al	b11+1...b1m
b21	a2	b23...b21	a1+1 ...am
al	a2	b33...b31	b31+1...b3m

Since we can always covert any row of the tableaux to contain all distinct symbols a1,a2,...,am, the three relations has lossless join property. For example, lets consider the last row

1) b33...b31 under X-YZ can be replaced by a3,...,al because Kj-->X-YZ holds in the original relation, and

2) b31+1...b3m under YZ can be replaced by a1+1...am because Kk-->YZ holds in the original relation.

5.5.2.4 Decomposition Examples.

Example 5.18

Given an E-set, E1(ABCDE)
with a minimum cover F1={A-->BC; C-->DE}.
An embedded object CDE is found,
decompose E1 into:

E2(AB) with F2={A-->B},
E3(CDE) with F3={C-->DE},
and R4(E2,E3:) with F4={A-->C}.

F1 is implied by F2 and F3 and F4 and F-maintenance is not required.

Example 5.19

Given an E-set, E0(ABCD)
with a minimum cover F0={A-->BD; B-->C; D-->C}.
An embedded object DC is found,
decompose E0 into:

E1(AB) with F1={A-->B},
E5(DC) with F5={D-->C},
and R6(E1,E5:) with F6={A-->D}.

F0 is not preserved after this decomposition,
the FD B-->C is not implied by F1 and F5 and F6.
We do the F-maintenance by adding the duplicate attribute C to E1. We now have,

E1(ABC) with F1={A-->B; B-->C},
E5(DC) with F5={D-->C},
and R6(E1,E5:) with F6={A-->D}.

F0 is now preserved by these three set,
but an embedded BC now appears in E1. So
E1 is decomposed to

E2(A),
E3(BC) with F3={B-->C},
and R4(E2,E3:) with F4={A-->B}.

E5 remains unchange as

E5(DC) with F5={D-->C}.

R6 is modified after E1 is replaced as

R6(E2,E5:) with F6={A-->D}.

Example 5.20

Given an E-set, E1(ABCDEF)
with F1={A-->BC; C-->DE; DE-->F}.
An embedded object CDEF is found and
E1 is decomposed into:

E2(AB) with F2={A-->B},

E3(CDEF) with F3={C-->DE; DE-->F},

and R4(E2,E3:) with F4={A-->C}.

F1 is preserved in E2, E3 and R4, but E3 contains an embedded object DEF. E3 is decomposed into

E31(C),

E32(DEF) with F32={DE-->F},

and R33(E31,E32:) with F33={C-->DE}.

F1 is preserved by F2, F4, F32 and F33.

5.5.3 Decomposition of a Composite E-set.

If an E-set has a composite key and this composite key is made up of several embedded object keys, then the E-set is called a composite E-set. And the embedded objects whose keys made up the composite key are called the component objects of the composite E-set. After decomposing the component objects, a special R-set may be used to tie all the component objects together with the composite E-set. An R-set used to tie a composite E-set with its component objects is called a "two-key" R-set. Furthermore, if the composite

E-set is not related to any other E-sets besides its component objects, then we may combine the composite E-set into the "two-key" R-set.

Let Ei(X1...Xn Y1...Yn Z) be an E-set.

If X1...Xn is the key of Ei and

X1Y1, X2Y2, ..., XnYn are embedded objects of Ei

then Ei may be decomposed into

Ej(Z),

Ej+1(X1Y1),

.

.

.

Ej+n(XnYn) and

Rk(Ej: Ej+1,...,Ej+n:). Rk is a special "two-key" R-set where Ej and Ej+1, ...,Ej+n both form the keys of Rk. In this R-set

card(Rk)=(m,1,1,...,1) and

card(Rk)=(1,m ,m ,...,m) are both true. In addition to this decomposition, if Ej is not related to anything else we may combine Ej into Rk. Rk now has all the attributes of Ej, .i.e.,

Rk(Ej+1,...,Ej+n: Z) and card(Rk) becomes (m ,m ,...,m).

Example 5.21

Let E1(S#,Mid,Wid,Wloc,Wphone,Mdesc,Qty) be an E-set, where

S# - stock number
Mid - merchandise id
Wid - warehouse id
Wloc - warehouse location

```
    Wphone - warehouse phone number
    Mdesc  - merchandise description
    Qty    - quantity on hand.
  Let F  = { S#-->Mid,Wid;
             Mid,Wid-=>S#;
             Mid-->Mdesc;
             Wid-->Wloc,Wphone }.

  We may decompose E1 into
    E2(S#,Qty),
    E3(Mid,Mdesc),
    E4(Wid,Wloc,Wphone), and
    R5(Stock: Merchandise,Warehouse:).
  If E2 is not related to any other R-set
  besides R5 then combine E2 and R5 and form
    R6(Merchandise,Warehouse: S#,Qty).
```

5.5.4 Problem Causes by Duplicate Attributes in Various E-sets.

As a result of F-maintenance process, the same general attribute may appear in several E-sets after a decomposition. If duplicate attributes are introduced in two or more E-sets, independent updates on these E-sets may violate some of the dependencies in the context of the original E-set even when all the dependencies are preserved. This problem is not indicative to the E-R model, the same type of problem exists in any database model. Ullman [Ul] did not address this problem in the relational model in his book. We will use an example to illustrate that there is an inherent problem associated with duplicate attribute in the extended E-R model. The same argument can be applied to the relational or any other database model.

Example 5.22

The following example shows that the FDs in the original E-

set is violated when independent updates are performed on the E-sets which are the result of some decompositions of the original E-set which require F-maintenance.

Let the original E-set and its minimum cover set be

E0(ABCD) and F0={A-->BD; B-->C; D-->C}.

An instance of E1 is shown in the following diagram.

E0

a1	b1	c1	d1
a2	b1	c1	d2
a3	b2	c2	d3

After the decompositions shown in the example 5.19 we have

E2(A),

E3(BC) with F3={B-->C},

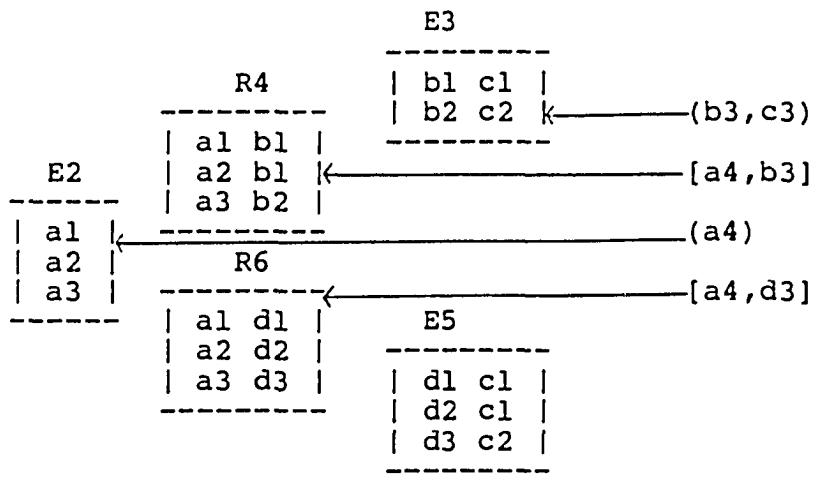
R4(E2,E3:) with R4={A-->B},

E5(DC) with F5={D-->C},

R6(E2,E5:) with F6={A-->D}.

F0 is preserved by F3, F4, F5 and F6.

The instances of these sets are show below.



In the database of example 5.22, we insert (a4) into E2, (b3,c3) into E3, and connect [a4,b3] in R4, and [a4,d3] in R6. Each one of these operation is valid (does not violate projected constraints on each set). But, when the entire structure is accessed together, we have two additional tuples (a4,b3,c2,d3) and (a4,b3,c3,d3) which violate the FDs in F0 (A is no longer a key of E0).

To remedy this problem, we must introduce explicit constraints to force a "join-link" checking on each independent update. For the database in example 5.22 the following explicit constraint is needed:

if [ai,bj] is in R4 & [ai,dk] is in R6 & (bj,cl) is in E3 & (dk,cm) is in E5 then cl = cm.

5.5.5 Embedded Object in an R-set as a Result of Transitive Dependency.

Let R1(E1,E2,...,En: XYZ) be an R-set and F1 be a minimum cover of this R-set. An embedded object is a set of attributes XY

where $Y \not\subseteq X$

and $XY \not\rightarrow XYZ$

and $X \rightarrow Y \in F1$.

(X-->Y is not a trivial FD and XY does not contain a key of R1). If the embedded object XY is detected in the R-set R1, we may decompose R1 into

R2(E1,E2,...,En,En+1: Z) and En+1(XY)

and set card(R2) to card(R1) $\cup$ {1}.

The FDs preserved by R2 and En+1 are

1) The Fds implied by card(R1) are preserved in a subset of card(R2),

2) the FD, X-->Y in En+1, and

3) the FD, E1,...,En-->En+1 in card(R2). The last FD preserves the dependency of X (and Y) on the key of E1.

When all of the FDs in the original R-set context are not preserved after a decomposition the F-maintenance process must be performed in the same manner as is done after an E-set decomposition. Again, the designer is recommended to decompose an embedded object whenever F-maintenance is not required. The embedded object, decomposed from an R-set, is merged into existing E-graph in the database if they represent the same entity type.

Example 5.23

Let R1(E1,E2: Warehouse,Addr,Phone,Qty) be an R-set,

E1(Supplier#,Name,Addr) and

E2(Part#,Partname,Desc) be E-sets.

Let card(R1) = (m,n) and

Warehouse-->Addr, Warehouse-->Phone are both in F1.

We may decompose R1 into

R2(E1,E2,E3: Qty) where card(R2)=(m,n,1) and

E3(Warehouse,Addr,Phone).

5.5.6 The FD Implied by a Connectivity Constraint of an R-set.

Given an R-set R1(E1,E2,...,El,El+1,...,En: X) and

card(R1)=(N1,...,Nl,Nl+1,...Nn).

If N1,...Nl > 1 and Nl+1,...,Nn = 1 then the FD

E1,...,El-->El+1,...,En holds in R1.

Proof:

Assuming the card(R1) is specified as shown above, but the FD

E1,...,El-->El+1,...,En does not hold in R1 then an instance of R1 may contain 2 tuples

t1=(a1,a2,...,al,...,ai,...,an, x1) and

t2=(a1,a2,...,al,...,bi,...,an, x2).

But the tuples t1 and t2 violate connectivity constraint of R1, i.e., the same set of entities a1,...,al is related to at most one entity drawn from Ei where l+1 <= i <= n. In other words, in any instance of R1, a unique set of entities drawn from E1,...,El is related to at most one entity drawn from Ei (l+1<=i<=n). So we may conclude that E1,...,El-->Ei (i=l+1,...,n) holds in R1, and by the union rule of the FDs E1,...,El-->El+1,...,En.

5.5.7 Collapsing E-sets.

Assuming that the FD El+1,...,En-->El,...,El exists in an R-set Rl as a result of card(Rl). If the E-sets El,...,El are one-node E-graphs and Rl is the only R-set in which El,...,El participate, then El,...,El may be eliminated from the database. And all of their general attributes may be combined with attributes of Rl.

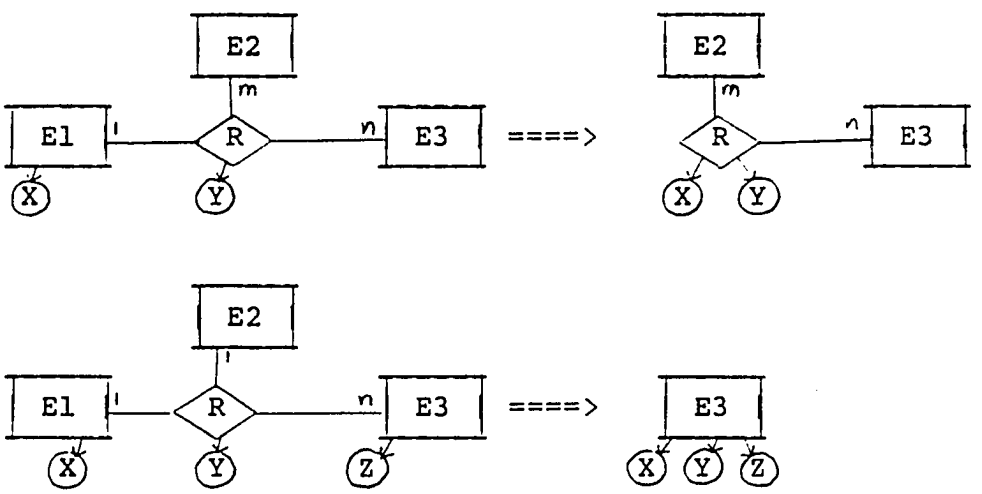
Let Rl(El,...,En) be an R-set and

El+1,...,En-->El,...,El holds in Rl. And El(Xl), E2(X2), ..., El(Xl) are the E-sets participating in only Rl and nothing else. Then eliminate El,...,El and modified

Rl(El,...,El,El+1,...,En: X) to

Rl(El+1,...,En: X Xl X2 ... Xl).

Example 5.26



The bottom diagram of example 5.26 shows a special case where only one E-set remain. In this case, the R-set is also eliminated from the database and all general attributes

are moved to the remaining E-set.

5.5.8 Composite Relation Decomposition.

When an MVD or an EMVD is found among E-sets participating in an R-set, the R-set is said to be a composite relation.

Let R1(E1,...,En:) be an R-set without any attributes and E1,...,El$\longrightarrow\!\!\!\!\!\rightarrow$El+1,...,Em be an MVD that holds in R1. We may decompose R1 into

R2(E1,...,El,El+1,...,Em:) and

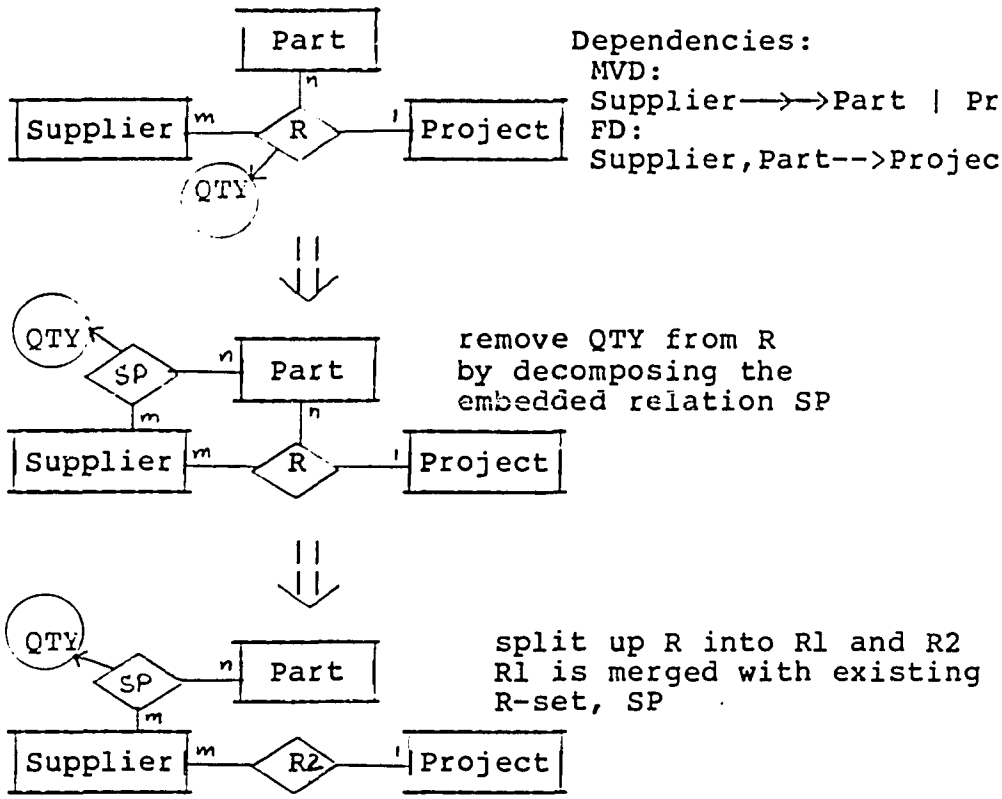
R3(E1,...,El,Em+1,...,En:). If R1 has some attributes then, we may have an EMVD among the E-sets. If these attributes may be eliminated from R1 by some means, then the EMVD becomes MVD and R1 may be decomposed into two R-sets as mentioned above.

Let R1(E1,...,En: X) be an R-set and

E1,...,El $\longrightarrow\!\!\!\!\!\rightarrow$ El+1,...,Em | Em+1,...,En

be an EMVD that holds in R1. If X can be removed from R1 by decompositions, then R1 may be decomposed into R2 and R3 as shown above.

Example 5.27



5.6 E-graph and R-graph Analysis.

Let G be an E-graph or an R-graph, and let g be an instantiation of G. The instantiation g is either an e-graph created by insertion operation or an r-graph created by connection operation. We also define two types of subgraph of G called road and detour.

5.6.1 Definitions:

A road R of a graph G is a subgraph of G which consists of several levels of nodes starting from the root node at level one. A road may be created from G by the following steps.

1) Include the root node of G into level 1 of R, and let level one be the current level of R.

2) Do road extension described in step 3) on the current level. If the road extension is successful, then repeat step 2) on the next level. Otherwise, terminate the road creation process.

3) Road extension on the current level of R is done by checking each node at the current level. If there is a category with onto mapping under each node, then include the arcs and nodes belonging to these categories at the next level of R. And the road is extended successfully.

4) If there is at least one node at the current level without any category with onto mapping under it, then the road can not be extended.

Since each road is created by following the categories with onto mapping starting from the root node of G, each instantiation g of G must include at least one node from each level of every road. Because any instantiation of E-graph or R-graph aways includes the root node and all the nodes accessible from the root through arcs with onto mapping constraint.

A detour D of a road R of a graph G is also a subgraph of G created from a set of all nodes at one level of R. The following steps may be used to create a detour D of a road R.

1) Let n be two.

2) Let nodes N1,N2,...,Nm be at level n of R.

3) For each node Ni (i=1,...,m) of R, find a path Pi from the root node to Ni, where Pi is not a path of R.

4) If we can find paths Pi (i=1,...,m) to all nodes Ni (i=1,...,m), then these paths form a detour of R. And the process is terminated successfully.

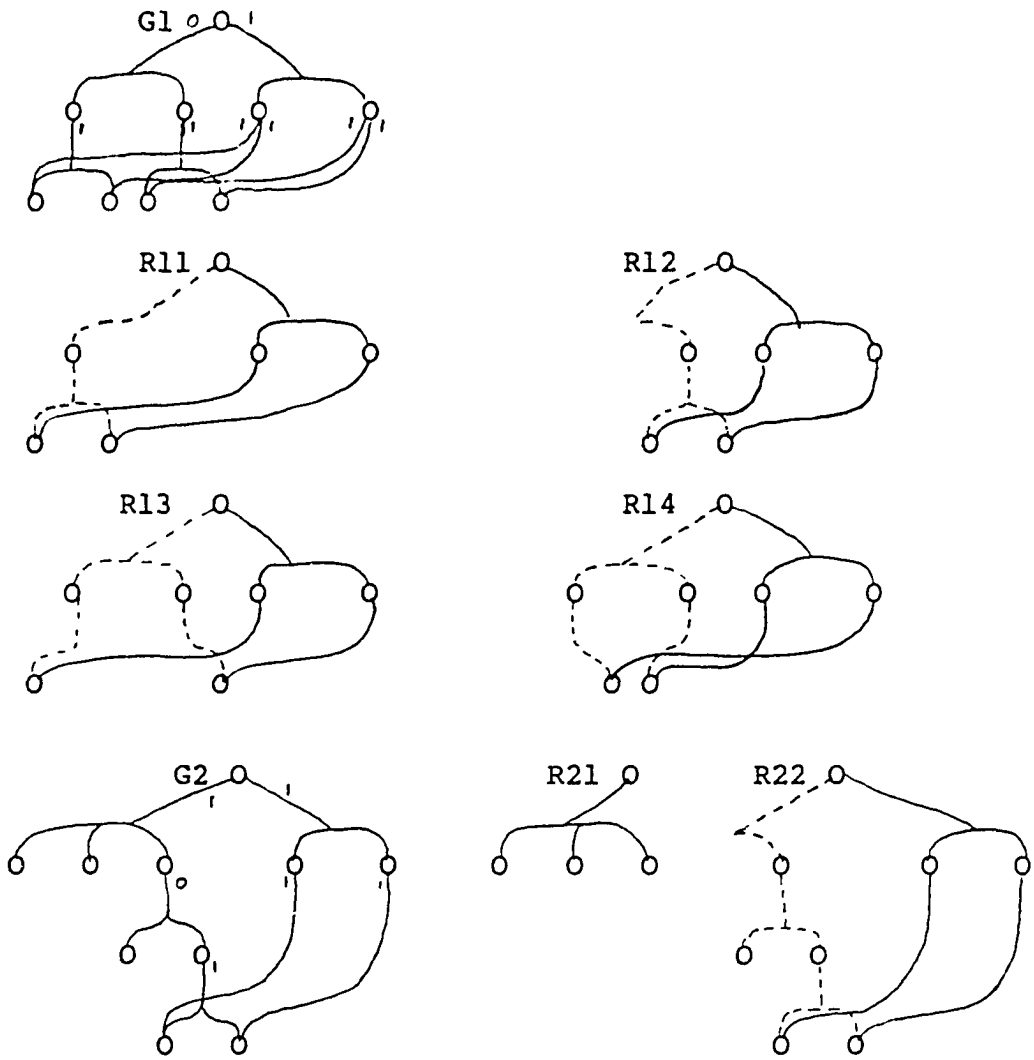
5) If n is the last level of R, then R has no detour and the process is terminated unsuccessfully.

6) If the paths to all nodes at level n of R can not be found, increment n by 1, and repeat the process from step 2.

According to the definition of a road, at least one node at level n of R must be included in each instantiation g of G. The generic structure of G requires that if a node of G is included in g, then all paths from the root of G to that node must also be included in g. Thus, any instantiation of G must include at least one root originated path from every detours of G.

Example 5.28 Graphs and their roads and detours.

a road ——— a detour - - - - - -



5.6.2 Detour Analysis Procedure.

Given a detour D of a road R of a graph G. The mapping constraints of various categories in D may be verified and some null nodes in G may be identified by the following procedure.

1) Let the root node be the current node.

2) If there are more than two categories or no category under the current node, then terminate the process.

3) If there is exactly one category under the current node, then check the splitting constraint of that category.

3.1) If the mapping constraint of the category is into, then change it to onto.

3.2) If the disjoint constraint of the category is disjoint, then any nodes under the category in G that are not part of D are null nodes.

4) If the current node has only one immediate successor in D, then repeat the process from step 2) with the successor node as the new current node.

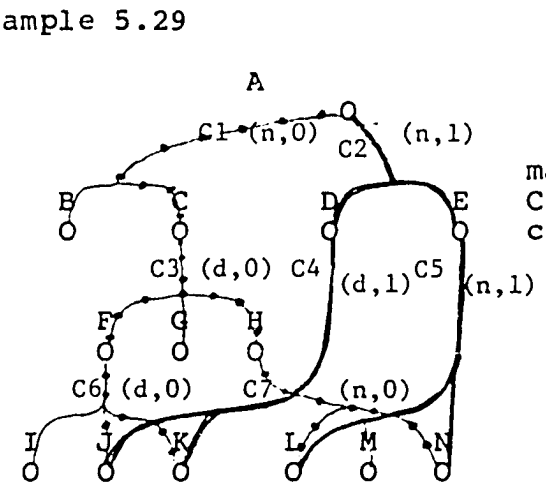
5) If the current node has several immediate successors in D, then check the disjoint constraint of the category of these nodes.

5.1) If the nodes are non-disjoint, then terminate the process.

5.2) Otherwise, make a successor node of the current node into the new current node and repeat the process from step 2) for each successor node in turn.

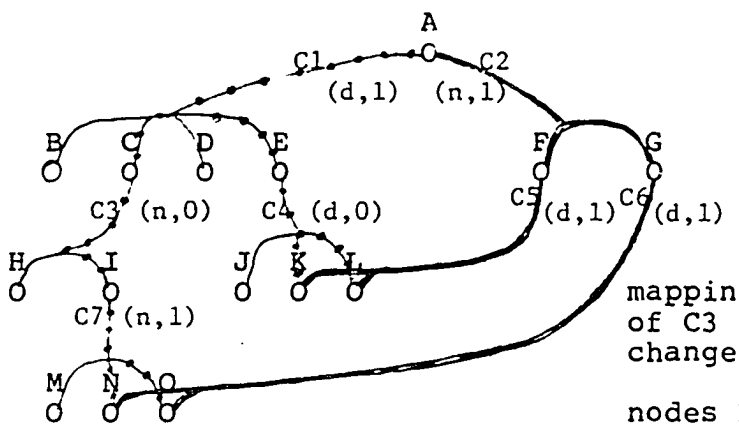
The proof of the detour analysis procedure is given in APPENDIX C.

Example 5.29



mapping constraints of C1, C3, C6 an C7 are changed to onto

nodes G and I are null.



mapping constraints of C3 and C4 are changed to onto

nodes B, D and J are null

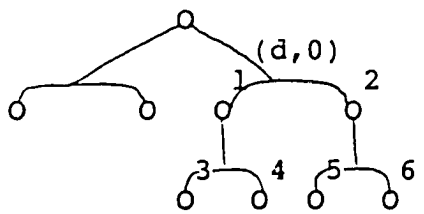
5.6.3 Null Subgraphs in a Graph G.

A null subgraph of G consists solely of one type of nodes called null nodes. If a null node is included in any instantiation, it will violate some splitting constraints of the graph G. So, a null node can never be instantiated. A null node may be identified in the detour analysis process. A null node is also a node in G which has more than one parent nodes which are disjoint.

Two nodes in G are disjoint if they are specified as

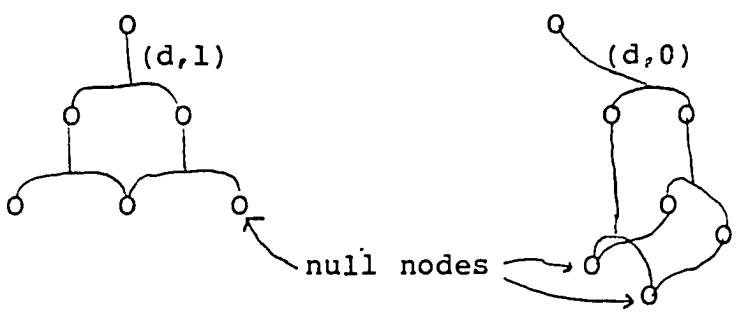
such under some category. Disjointness is also inherited by successors, i.e., if two nodes are found to be disjoint then their children are also disjoint. The following example illustrates this point.

Example 5.30



because 1 and 2 are disjoint,
==> 1 and 5 are disjoint
==> 1 and 6 are disjoint
==> 2 and 3 are disjoint
==> 2 and 4 are disjoint
==> 3 and 5 are disjoint
==> 3 and 6 are disjoint
==> 4 and 5 are disjoint
==> 4 and 6 are disjoint

The next example shows the null nodes whose parents are disjoint.

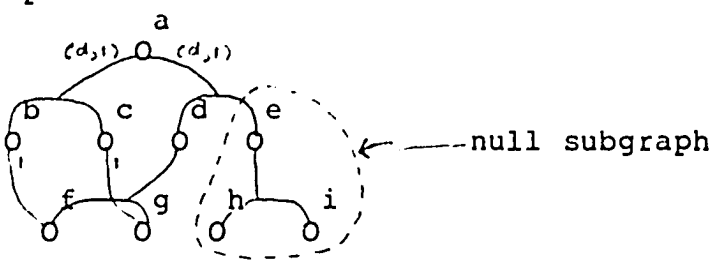


After all the null nodes are identified in a graph G, we may create null subgraphs by the following rules.

1) If all of the sibling nodes under a category are null nodes and the mapping constraint of the category is onto, then the parent node is a null node.

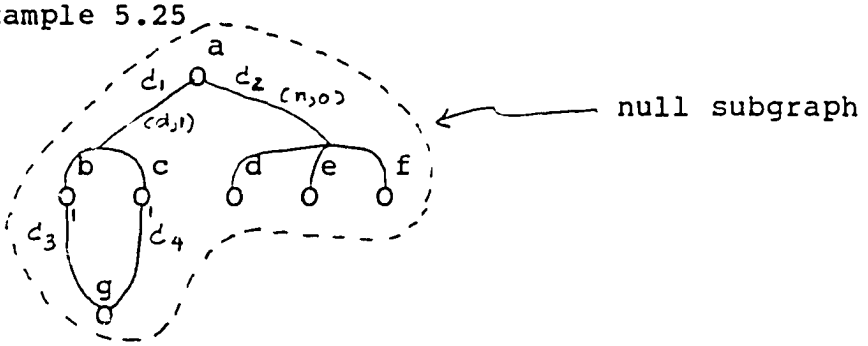
2) If a node is a null node, then all of its children nodes are null nodes.

Example 5.24



In example 5.24, the node e is identified as a null node by the detour analysis. And because the parent node e is a null node, h and i are null nodes.

Example 5.25



In example 5.25, g is a null node because its parents, a and c, are disjoint. And b and c are null nodes because their only child under the categories C3 and C4 is a null node. And a is a null node because its children under the category C1, whose mapping constraint is onto, are null nodes. And finally, d, e, and f are null nodes because their parent a is a null node.

If a null subgraph is identified in G, the designer must make the correction by

- eliminate the subgraph from G, or
- change the mapping constraints and/or disjointness

constraints that cause the nodes to be null.

This correction is done on the trial and error basis, further study is required for more systematic approach.

5.7) Summary.

In this chapter, a top down design technique for extended E-R databases is presented. This technique can be divided into four major steps as follow.

1) Object identification step - in this step, the designer collects the users' requirements and identifies the E-sets, the R-sets and their attributes to be used in a database.

2) E-graphs and R-graphs creation step - after identifying all the sets, the designer form different E-graphs and R-graphs from these sets. An E-graph (R-graph) is formed by selecting a E-set (R-set) as a root node or by selecting several E-sets (R-sets) as immediate successers of a new root node. Each node in a graph is examined from the root node down. A node in a graph is specialized if the designer detects inapplicable attributes or embedded FDs in the node. A node could also be specialized according to some application requirements. As the result of a node specialization, the graph in question is extended by adding a set of nodes which belong to the same category under the original node. The categorization process is used as quidelines in a node specialization.

After an E-graph ar an R-graph is fully extended by node specializations, the designer perform normalization on each node starting from the root and work downward level by level. For each node, the FDs among the attributes of the node are examined. If embedded objects are found, they are projected out to form new E-graph or R-graphs in the database. This process is simila to the relational decomposition technique except that when an embedded object is removed, it still ties to the original node by an R-set. For a node in an R-graph, the designer also looks for an MVD or an EMVD among the key attributes. If one is found, the R-graph may be splitted into two graphs.

3) Splitting constraints assignment step - the graphs created in the previous step are assigned splitting constraints in this step. For each category under each node in a graph assign a splitting constraint based on the category type and the underlying semantics.

4) Graphs analysis step - at this point, an E-graph or an R-graph with its splitting constraints represents a unique entity or relationship concept in the database. The root node is the primary role of that concept. The other nodes in the graph represent secondary roles of the same concept. A splitting constraint specifies the type of relationship among different roles. It is possible for the designer to assign conflicting constraints to an E-graph or an R-graph. If this is the case, some roles are isolated from the primary role and are not part of the logical struc-

ture of the graph. The graph analysis process uses an algorithm to identify the nodes which are isolated as a result of conflicting constraints. The algorithm identifies subgraphs called roads and detours based on splitting constraints of different categories starting form under the root node. The designer may use this algorithm to identify and correct conflicting constraints in a graph and make all nodes part of the logical structure of a single concept.

CHAPTER SIX

CONCLUSION, RECOMMENDATIONS AND FUTURE WORKS

6.1 Conclusion.

We have introduced the extended E-R model in this work. This model may be used as a designing tool. Usually, a designer organizes the logical structure of a real world system into a conceptual schema in the extended E-R model, then converts this schema into some existing model like hierarchical, network, or relational models. The extended E-R model can also be used as an actual database model. In this case, an extended E-R database is implemented on other existing database system.

The extended E-R model maintains the basic constructs of the standard E-R model, i.e., E-sets, R-sets, V-sets, roles, and attributes. The extended model also includes the following structural constraints of the original E-R model: set membership constraints, connectivity constraints, and existency constraints of weak entity sets. In addition, the extended model provides three generalization structures for the E-sets, R-sets, and V-sets. These structures form three directed graphs whose nodes and edges represent sets and generalizations among sets. E-sets and R-sets generalization structures are labeled graphs because an E-set or R-set may be specialized more than once under different categories

(labels). The structure of E-sets is partitioned into several subgraphs called E-graphs. Each E-graph represents a unique entity type in a database. The structure of R-sets is similarly partitioned into different R-graphs.

There are additional types of structural constraints in the extended model. Associated with each category under a node in an E-graph (R-graph) is an E-splitting constraint (R-splitting constraint). An E-splitting constraint (R-splitting constraint) indicates the type of mapping between a generic set and its specialized sets and the disjointness of the specialized sets under the category. A new class of attributes called type specific attribute is introduced in the model. This type of attribute represents a property of an E-set or R-set and not the definition of properties of members of the set.

Additional structures of the extended E-R model allow a designer to build more semantic information into the conceptual schema of a database without having to express this information as explicit constraints. In order to visualize the conceptual schema of a database, a designer may create an extended E-R diagram. This diagram is an expansion of the E-R diagram that includes the new structures of the extended model. A diagram presentation technique is presented base on the generalization of the diagram itself. Using this technique, a diagram is considered to have multiple layers. Each layer consists of a set of objects each of which is at a certain level of generalization abstraction.

A few consecutive layers, called the current view, are displayed to the users of the database at one time. By viewing the conceptual schema of a database through the current view, the users can concentrate on a particular part or aspect of a database while ignoring the rest. The users are also provided with schema operators like zoom-in, zoom-out, show-attr, omit-attr,..etc. These operators may be used to modified the current view and to explore different parts of the database schema.

Finally, a top down designing technique for the extended E-R database is presented. After gathering information, the E-sets, R-sets, attributes, and V-sets are identified. Then different E-graphs and R-graphs are created from the E-sets and R-sets identified. For each E-graph and R-graph, a node may be specialized based on embedded FDs, inapplicable attributes, or application requirements. After the specialization process, embedded objects may be decomposed from each node of an E-graph or R-graph based on FDs and MVDs in that node. After the decomposition process each E-graph and R-graph is analyzed, then empty sets (null nodes) are eliminated from the graph. An empty set in a graph is the result of conflicting structural constraints of that graph.

6.2 Recommended Implementation.

The extended E-R database may be implemented by any existing database system. However, relational database sys-

tem, particularly those with system-controlled key attributes, are the most suitable systems. Two examples of this type of database systems are the RM/T system [C8] and the TAXIS system [M2]. It is recommended that the extended E-R database should be implemented on a database system with the following properties:

1) The system is relational.

2) Each relation in a database of the system has a system-key attribute controlled by the DBMS.

3) The values of the system-key attributes are transparent to the users.

4) The users are not allowed to modifies the values of the system-key attributes.

Following the technique proposed in [C6], the basic constructs of the E-R database are represented by different types of relations in the relational database system. An E-set is implemented by an E-relation. The attributes and their domains of the E-relation schema represent the general attributes and their V-sets of the E-set. A tuple in the E-relation represents an entity and its attributes' values. The general attributes shared by several E-sets are not stored redundantly in every E-set. Each general attributes is stored once in the most general E-set.

An R-set is implemented by an R-relation. The roles and E-sets of the R-set are represented by the special attributes of the R-relation schema. These attributes have the same domain called E-relation domain. The general at-

tributes and their V-sets of the R-sets are represented by other attributes and domains of the R-relation schema. A relationship of the R-set is represented by a tuple in the R-relation. This tuple has a composited key made up of system-key values of the tuples representing entities participating in the relationship.

Each additional construct of the extended E-R model is implemented by a special relation in the relational system. The generalization structure of V-sets is represented by the GV-relation. A tuple of this relation is a pair of domain names representing a generic V-set and a specialized V-set. The type specific attributes are represented by the TS-relation. A tuple in this relation is a triplets of an E-relation or R-relation name, a value type, and a value or a database procedure name (for virtual value). The generalization structure of E-sets is represented by the GE-relation. Each tuple in the relation is a triplet of a generic E-relation name, a category, and a specialized E-relation name. The E-splitting constraints are represented by the ES-relation whose tuple contains a generic E-relation name, a category, a mapping constraint, and a disjoint constraint. Similarly, the generalization structure of R-sets and R-splitting constraints are represented by the GR-relation and RS-relation respectively.

The information stored in these special relations is required by schema operations to assemble the current view. The same information is also required for some database

operations. For example, an insert or delete operation requires that an e-graph be created from the GE-relation and various E-relations. For a connect or disconnect operation, an r-graph must be created from the GR-relation and some R-relations. These operations should be done by database procedures written at database generation time.

6.3 Future Works.

There are several aspects of the extended E-R model which may be improved by further research. In what follow, three of these aspects are presented.

The current insert and delete algorithms only allow an e-graph to be created into a database or deleted entirely from a database. These two algorithms can be modified to allow adding and deleting nodes from an existing e-graph. An e-graph of this type is more suitable for representing a real world entity which evolves over time. Consequently, an e-graph depicts an object playing certain roles and the object may change roles during its lifetime.

In the graph analysis process, the null subgraph elimination is done on a trial and error basis. This process is not practical for a graph with a large number of nodes. A good heuristic algorithm with reasonable respond time should be developed to isolate and correct null subgraphs of an E-graph or R-graph.

Finally, the comparison of the performance of the extended and standard E-R model should be investigated. The

empirical comparison can be obtained by creating two pilot databases of the two model from the same set of data. A set of transactions is processed against both databases. The performance of each database is quantitatively evaluated and compared to each other.

REFERENCES

[A1] Aho, A.V., Beeri, C. and Ullman, J.D. "The Theory of Joins in Relational Databases." ACM TODS 4,3 (Sept 79), pp. 297-314.

[B1] Batini, C. and Santucci, G., "Top-Down Design in the Entity-Relationship Model." Entity-Relationship Approach to System Analysis and Design, North-Holland, Amsterdam, 1980, pp. 323-338.

[B2] Beeri, C., Fagin, R. and Howard, J.H. "A Complete Axiomatization for Functional and Multivalued Dependencies in Database Relations." ACM-SIGMOD 77 Int. Conf. on Management of Data, Toronto, August 77, pp. 47-61.

[B3] Beneworth, R.L., Bishop, C.D., Turnbull, C.J., Holman, W.D. and Monette, F.M. "The Implementation of GERM, an Entity-Relationship Data Base Management System." Proc 7th VLDB Conf. (1981), pp. 478-485.

[B4] Bernstein, P.A. and Goodman, N. "What Does BCNF Do?" Proc 6th VLDB Conf. (1980), pp. 245-259.

[B5] Biskup, J., Dayal, U. and Bernstein, P. "Synthesizing Independent Data Base Schemas." ACM-SIGMOD 77 Int. Conf. on Management of Data, Toronto, August 77, pp. 143-151.

[B6] Borgida, A. and Wong, H.K.T. "Data Model and Data Manipulation Languages: Complementary Semantics and Proof." Proc 7th VLDB Conf. (1981), pp. 260-271.

[B7] Borkin, S.A. Data Model : A Semantic Approach for Database System, The MIT Press, Cambridge, Mass, 1980.

[B8] Brodie, M.L. "On Modeling Behavioral Semantic of Databases." Proc 7th VLDB Conf. (1981), pp. 32-42.

[C1] Cammarata, S. "Deferring Updates in a Relational Data Base System." Proc 7th VLDB Conf. (1981), pp. 286-293.

[C2] Chan, E.P.F. and Lochovsky, F.H. "A Graphical Data Base Design Aid Using the Entity-Relationship Model." Entity-Relationship Approach to System Analysis and Design, North-Holland, Amsterdam, 1980, pp. 295-310.

[C3] "Chen, P.P. The Entity-Relationship Model: Toward a Unified View of Data." ACM TODS 1,1 (March 1976), pp. 9-36.

[C4] Chen, P.P. "The Entity-Relationship Model : A Basis for Enterprise View of Data." Tutorial : Centralized and Distributed Database System, IEEE Computer Society, New York, 1979 pp. 158-165.

[C5] Chen, P.P. "The Entity-Relationship Model : Toward a Unified View of Data." Tutorial : Centralized and Distributed Database System, IEEE Computer Society, New York, 1979, pp.166-193.

[C6] Chen, P.P. and Yau, S.B. "Design and Performance Tools for Database System." Tutorial : Centralized and Distributed Database System, IEEE Computer Society, New York, 1979, pp.222-234.

[C7] Codd, E.F. "A Relational Model of Data for Large Shared Data Banks." CACM 13, No 6 (June 75), pp. 377-387.

[C8] Codd, E.F. "Extending the Database Relational Model to Capture More Meaning." ACM TODS 4,4 (Dec 79), pp. 397-434.

[D1] Date, C.J. A Introduction to Database Systems (2nd Edition), Addison-Wesley Publishing Company, Reading, Mass, 1975, pp. 51-68.

[D2] Dos Santos, C.S., Neuhold, E.J. and Furtado, A.L. "A Data Type Approach to the Entity-Relationship Model." Entity-Relationship Approach to System Analysis and Design, North-Holland, Amsterdam, 1980, pp. 103-120.

[G1] Gallaire, H., Minker J. and Nicolas, J.M. "An Overview and Introduction to Logic and Databases." (Gallaire, H. and Minker, J. eds.) Logic and Databases, Prenum Press, New York, 1978, pp. 3-30.

[G2] Gardarin, G. and Bihan, J.L. "An Approach Towards a Virtual Data Base Protocol for Computer Networks." Tutorial : Centralized and Distributed Database System, IEEE Computer Society, New York, 1979, pp.465-475.

[H1] Hainaut, J.L. "Theoretical and Practical Tools for Data Base Design." Proc 7th VLDB Conf. (1981), pp. 216-224.

[H2] Hammer, M. and McLeod, D. "Database Description with SDM : A Semantic Database Model." ACM TODS 6,3 (Sept 1981), pp. 351-386.

[H3] Harary, F. Graph Theory, Addison-Wesley Publishing Company, Reading, Mass, 1972.

[H4] Hoare, C.A.R. "Notes on Data Structuring," in Structured Programming, (Dahl, O.J., Dijkstra, E.W. and Hoare, C.A.R. eds.) Academic Press, New York, 1972, pp. 83-174.

[K1] Kapp, D. and Leben, J.F. IMS Programming Techniques, Van Nostrand Reinhold Company, New York, 1978, pp. 1-5.

[K2] Kent, W. "Data Model Theory Meets a Practical Application." Proc 7th VLDB Conf. (1981), pp. 13-22.

[K3] Kent, W. "Consequence of Assuming a Universal Relation." ACM TODS 6,4 (Dec 81), pp. 539-556.

[L1] Lien, Y.E. and Ying, J.H. "Design of a Distributed Entity-Relationship Database System." Tutorial : Centralized and Distributed Database System, IEEE Computer Society, New York, 1979, pp. 476-480.

[L2] Lien, Y.E., Shapiro, J.E. and Tsur, S. "DSIS - A Data Base System with Interrelational Semantics." Proc 7th VLDB Conf. (1981), pp. 465-477.

[M1] Mc Leod, D. and King, R. "Applying a Semantic Database Model." Entity-Relationship Approach to System Analysis and Design, North-Holland, Amsterdam, 1980, pp. 193-210.

[M2] Mylopoulous, J. and Wong H.K.T. "Some features of the TAXIS Data Model." Proc 6th VLDB Conf. (1980), pp. 399-410.

[O1] Olle, T.W. The CODASYL Approach to Data Base Management, John Wiley & Sons, Ltd., 1978, pp. 7-40.

[P1] Poonen, H. "CLEAR : A Conceptual Language for Entities and Relationships." Tutorial : Centralized and Distributed Database System, IEEE Computer Society, New York, 1979, pp.194-215.

[R1] Reiter, R. "On Closed World Data Bases." (Gallaire, H. and Minker, J. eds.) Logic and Databases, Prenum Press, New York, 1978, pp. 55-76.

[R2] Roussopoulous, N. and Mylopoulous, J. "Using Semantic Networks for Database Management." Proc. 1st VLBD Conf. (1975), pp. 144-172.

[S1] Sakai, H. "On the Optimization of Entity-Relationship Model." Tutorial : Centralized and Distributed Database System, IEEE Computer Society, New York, 1979, pp.216-220.

[S2] Sakai, H. "A Method for Defining Information Structure and Transaction in Conceptual Schema Design." Proc 7th VLDB Conf. (1981), pp. 225-234.

[S3] Scheuermann, P., Schiffner, G. and Weber, H. "Abstraction Capabilities and Invariant Properties Modeling Within The Entity-Relationship Approach." Entity-Relationship Approach to Systems Analysis and Design, North-Holland, Amsterdam, 1980, pp. 121-140.

[S4] Smith, J.M. "A Normal Form for Abstract Syntax." Proc 4th VLDB Conf. (1978), pp. 156-162.

[S5] Smith, J.M. and Smith, D.P. "Database Abstractions : Aggregation." CACM 20,6 (June 1977), pp. 405-413.

[S6] Smith, J.M. and Smith, D.P. "Database Abstraction : Aggregation and Generalization." ACM TODS 2,2 (June 1977), pp. 105-133.

[S7] Stanat, D.F. and McAllister, D.F. Discrete Mathematics in Computer Science, Prentice-Hall Inc., Englewood Cliff, New Jersey, 1977.

[T1] Tsichritzis, C.D. and Frederick, H.L. Data Models, Prentice-Hall Inc., Englewood Cliff, New Jersey, 1982, pp. 16-36.

[U1] Ullman, J.D. Principles of Database Systems, Computer Science Press, Potomac, Maryland, 1980, pp. 1-10.

[W1] Wiederhold, G. Database Design, McGraw-Hill Book Company, 1979, pp. 368-399.

APPENDIX A

INSERT, DELETE, CONNECT AND DISCONNECT ALGORITHMS

Primitive Functions

The following primitive functions are defined on a directed labeled graph with the properties:

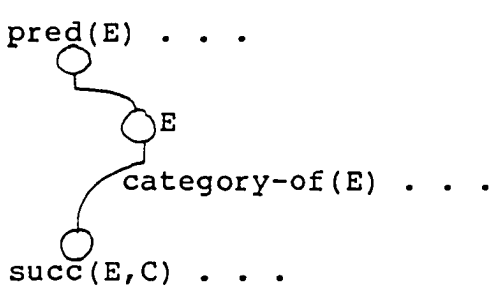
1) The graph is acyclic.

2) It has a distinct node whose indegree is zero.

3) Associated with each label under a node in the graph is a mapping and a disjoint constraints.

```
root-of(N) - return the root node of node N.
has-pred(N) - return true if N has a predecessor.
has-succ(N) - return true if N has a successor.
mapping-of(N,C) - return the mapping contraint
                  of C under N.
disjoint-of(N,C) - return the disjoint constraint
                   of C under N.
pred(N) - return successive predecessor of N.
succ(N,C) - return successive successor of N with label C.
category-of(N) - return successive label of N.
```

The general structure of E-graph with node E.



```
insert(e,G) /*insert e into a group of E-sets, G */
{
   Let G = {E1,E2,...,En} where
      E1,E2,...,En are siblings;
   if e in root-of(E1) then return;
   else
      for i <- 1 to n do create(e,Ei);
 }


create(e,E) /*create an e-node from E-node*/
{
   if e in E then return;
   if has-pred(E) then /*check contraints*/
   for pE <- pred(E) do
      if violate-constr(e,pE,E) then
                           abort-insert(e,E);
   E <- E ⋃ {e}; /*create a node here*/
   if has-pred(E) then /*propagate upward*/
      for pE <- pred(E) do create(e,pE);
   if has-succ(E) then /*propagate downward*/
      for C <- category-of(E) do
         if mapping-of(E,C) = "1" then
         {
            if exist-in-succ(e,E,C) then break;
            for wE <- which-one(E,C) do create(e,wE);
         }
}


delete(e,E) /*delete an entity e from E-sets E*/
{
   E0 <- root-of(E);
   if e not in E0 then return;
   else erase(e,E0);
}


erase(e,E) /*remove an e-node, e from an e-graph*/
{
   if e not in E then return;
   else {
      for each R(b1/E1,...,bn/En) such that
               E in {E1,...,En} do
         for each r in R do /*disconnect relationship*/
            if e in r then R <- R - {r}; /*involved e*/
      E <- E - {e}; /*remove e from E here*/
   }
   if has-succ(E) then /*propagate downward*/
      for C <- category-of(E) do
         for sE <- succ(E,C) do erase(e,sE);
}
```

Additional functions used in insert and delete.

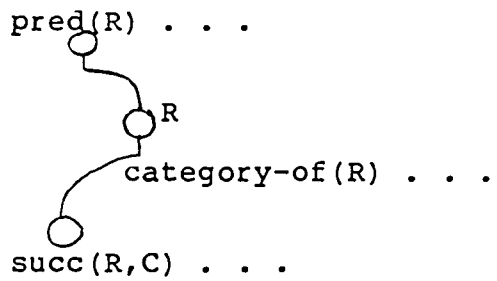
abort-insert(e,E) - terminate and remove partially built e-graph.

violate-constr(e,pE,E) - if disjoint constraint of an arc between pE and E indicates disjointness and e already exists in a sibling of E, then return true; otherwise, return false.

exist-in-succ(e,E,C) - if any successor of E under C has e as a member then return true, else return false.

which-one(E,C) - return successive successor of E under C that will be used in the insertion process.

The general structure of R-graph with node R.



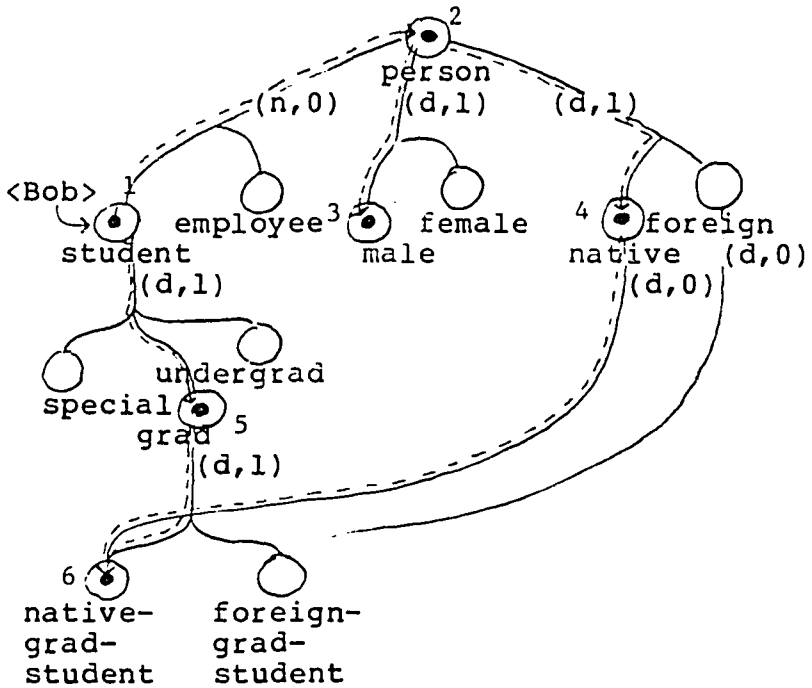
```
connect(r,R) /*create an r-graph with node r
                          from an R-graph of R*/
{
   if r in R then return; /*create new r-node here*/
      else if card(r,R) then R <- R ⋃ {r};
           else abort-connect(r,R);
   if has-succ(R) then /*propagate downward*/
      for C <- category-of(R) do
      {
         flag <- 0;
         for sR <- succ(R,C) do
         {
            Let r = (e1,...,en);
            Let sR = (b1/E1,...,bn/En);
            if e1 in E1 & ... & en in En then {
               connect(r,sR); flag <- flag+1;
            }
         }
         if mapping-of(R,C) = "1" and flag = 0
            then abort-connect(r,R);
         if disjoint-of(R,C) = "d" and flag > 1
            then abort-connect(r,R);
      }
   if has-pred(R) then /*propagate upward*/
      for pR <- pred(R) do connect(r,pR);
}


disconnect(r,R) /*erase an r-graph whose r-node is r*/
{
   if r in R then R <- R - {r}; /*remove r-node here*/
   else return;
   if has-succ(R) then /*propagate downward*/
      for C <- category-of(R) do
         for sR <- succ(R,C) do disconnect(r,sR);
   if has-pred(R) then /*propagate upward*/
      for pR <- pred(R) do disconnect(r,pR);
}
```

Additional functions used in connect and disconnect.

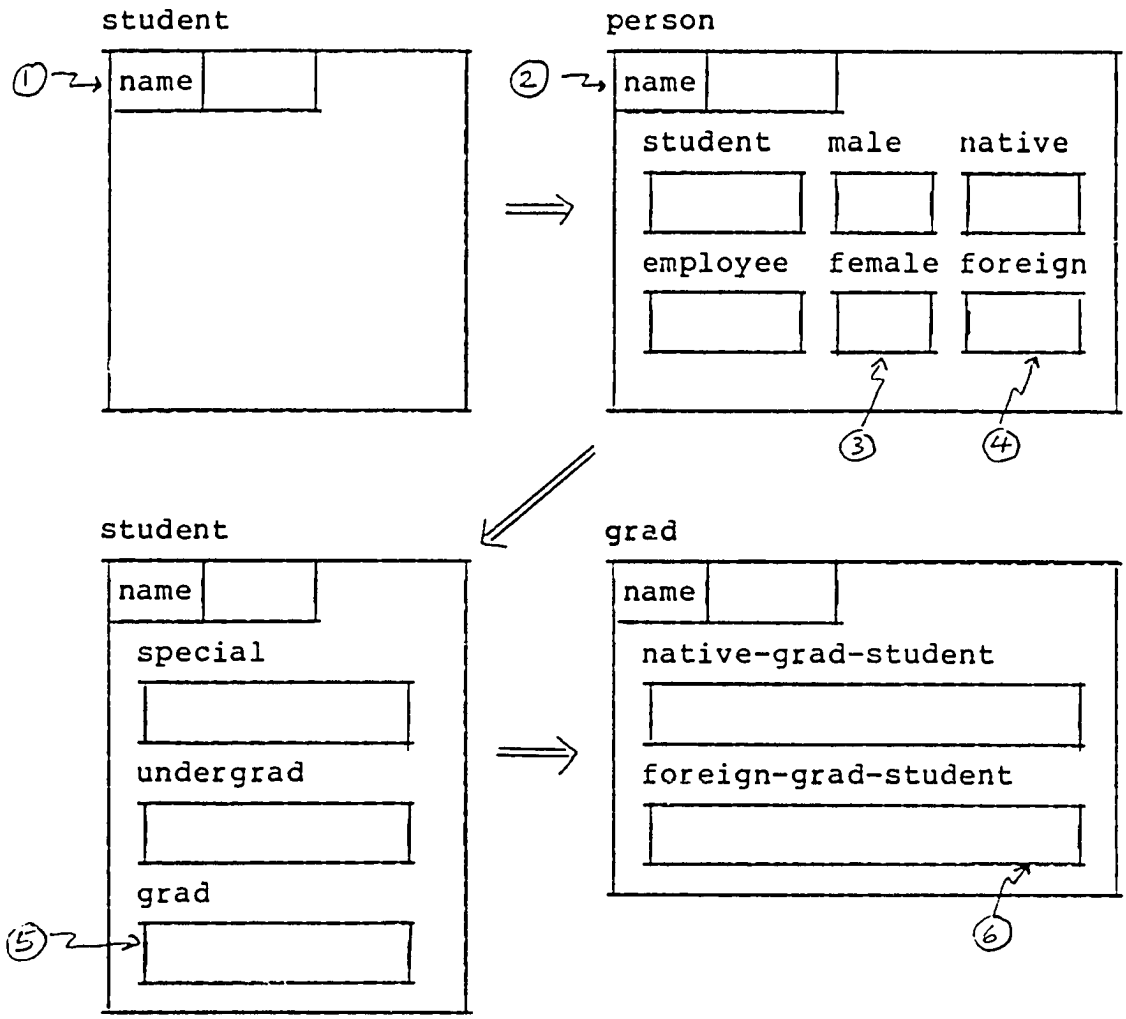
abort-connect(r,R) - terminate and remove partially built r-graph.

card(r,R) - if the relationship r satisfies the cardinality constraint of the R-set, R then return true otherwise return false.

The following diagrams illustrate the insert operation of an entity "Bob" into an E-sets, student.



The same operation may be shown as a sequence of different extended diagram views. The current view may be changed as the operation step from one E-set to another. The function which-one(E,C) shows all immediate successors of a current E-set under category C and prompts the user to select one or more of these E-sets.


student
person
name
name
student
male
native
employee
female
foreign
student
grad
name
name
special
native-grad-student
undergrad
foreign-grad-student
grad
1
2
3
4
5
6

APPENDIX B

MODIFYING R-SPLITTING CONSTRAINTS

An R-splitting constraint of a particular R-set is derived from the E-splitting constraints of all the E-sets which induce the R-set. The guidelines used to determine an R-splitting constraint were given in chapter 4. In some cases, after an R-splitting constraint is determined, it may be modified to express additional semantic of the R-set. The following rules can be used to modified an R-splitting constraint:

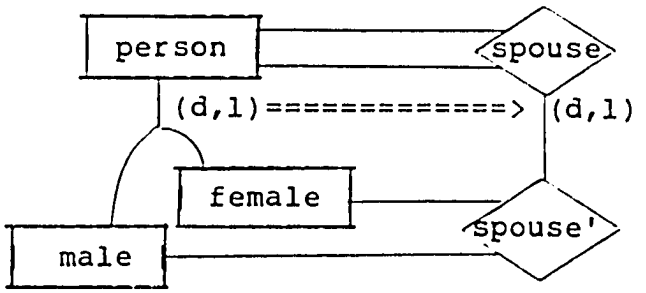
1) If the disjoint constraint is "d" then the constraint can not be modified.

2) If the mapping constraint is "1" then the constraint may not be modified.

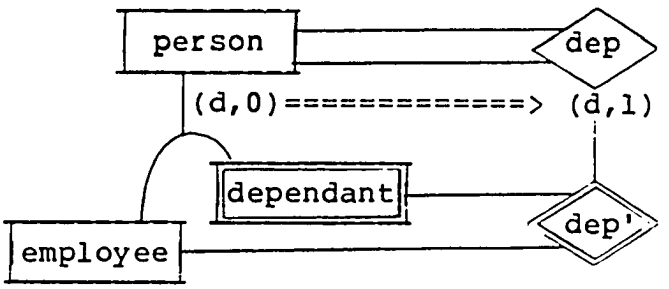
3) If the disjoint constraint is "n" it may be modified to "d" to force each relationship in the R-set which may be propagated downward to be in only one of the immediate successors.

4) If the mapping constraint is "0" it may be changed to "1" to force each relationship in the R-set to be the result of some relationship in some of its immediate successors.

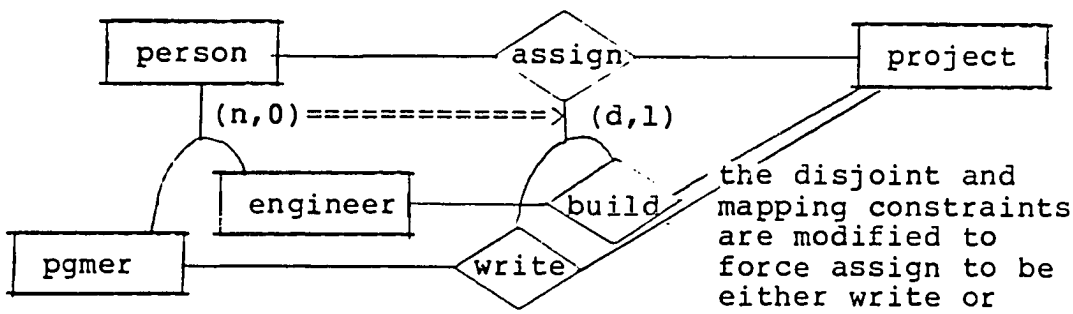
The following example show the modified R-splitting constraints of various R-sets.



this R-splitting constraint is implied by the E-splitting constraint and may not be modified



the mapping constraints is modified to make dep the same as the weak R-set dep'



the disjoint and mapping constraints are modified to force assign to be either write or build but not both

Relationship of Structural Constraints Between a Generic R-set and Its Specialized R-set.

A generic R-set should have less stringent constraint than its specialized R-sets. More stringent constraint of a specialized E-set may be expressed as a combination of

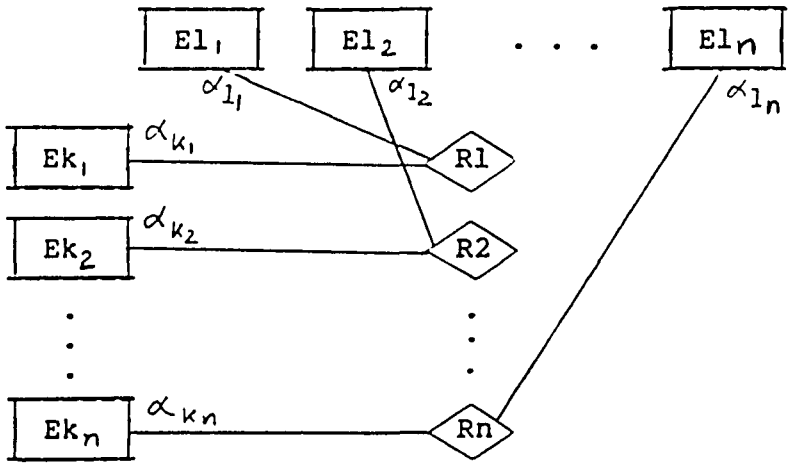
a) connections to more specific E-sets and

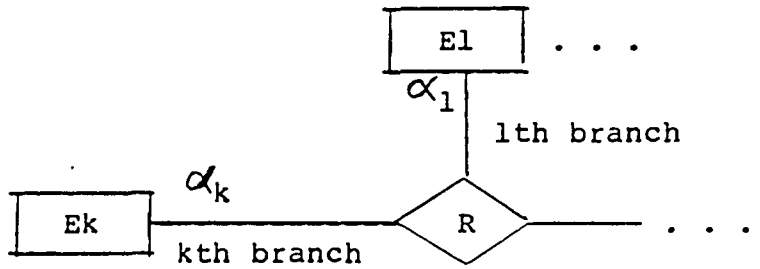
b) smaller maximum cardinalities.

This follow from the concept of specialization abstraction. If an R-set Rj is a successor of an R-set R, then a connection in Rj is actually a specialized connection in R (in term of specialized entities it connect to). So when a relationship is made to belong to Rj, we should be able to propagate this relationship through all predecessors of Rj including R.

A designer may use the algorithm given below to find a maximum cardinality of a particular branch of a generic R-set R given an R-graph that contain R as a node. The following diagrams depict the relation between the R-set R and its specialized R-sets.

Part of an R-graph, let R be an R-set among Ei, i=1,...,m; let the immediate successors of R under the category C be Rj, j=1,...,n.



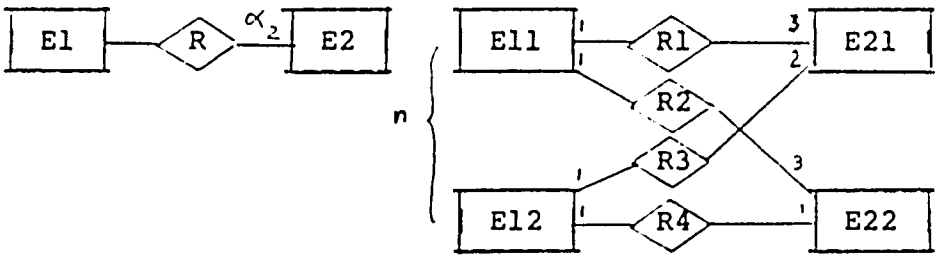


$Ei_1, Ei_2, \ldots, Ei_n$ are not necessarily distinct, and α_i is the maximum cardinality of Ei in R.

Algorithm to find appropriate α_j so that any relationship in Rj, j=1,...,n can be propagated to R. Assuming that R1,...,Rn are induced from R by E1,...,El. And Ei (i=1,...,l) has $Ei_1, Ei_2, \ldots, Ei_n$ as immediate successors under category Ci.

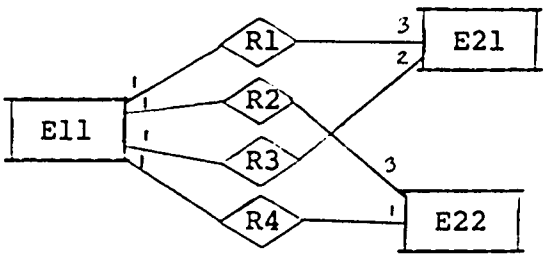
```
αj <- 0; for i <- 1,2,...,m where i ≠ j do {
   if i > 1 then { Ei1 <- Ei; n <- 1; }
   else {
        let n be the number of immediate successors
            of Ei under Ci;
        if Ei1,Ei2,...,Ein are nondisjoint then {
            modify the R-graph by replacing Ei1,Ei2,...,Ein
            with a single node Ei1 and attach existing
            ith branches of all Rj's to this node;
            n <- 1;
        }
        else leave Ei1,Ei2,...,Ein the way they are;
   }

   combine all R-sets, Rj1,Rj2,...,Rjp that are not
      distinguishable from each other into a single R-set
      called Rj1, assign the largest jth branch maximum
      cardinalities among these R-sets to the new R-set;

   for each Eik, k=1,...,n
                        /*each one of the ith branch node*/
        add up the jth branch maximum cardinality of all
        R-nodes incident on Eik and call it αjk;
   for k <- 1,...,n do
        if αj < αjk then αj <- αjk; }
```

Example 1 - find the maximum cardinality of E2 in R (α_2).

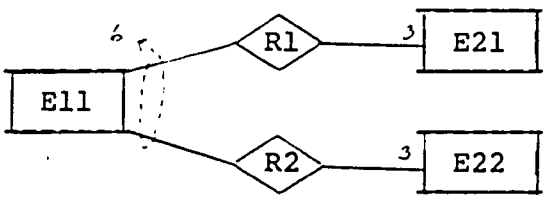


original R-graph (generic set)

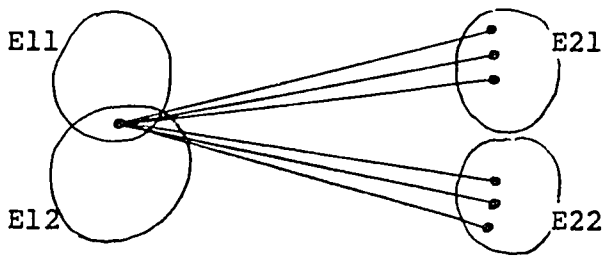
(specialized sets)



combine E-nodes E11 and E12

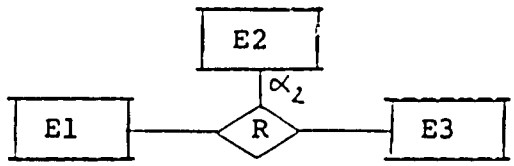


combine R-nodes and
add up cardinalities, $\alpha_2 = 6$.

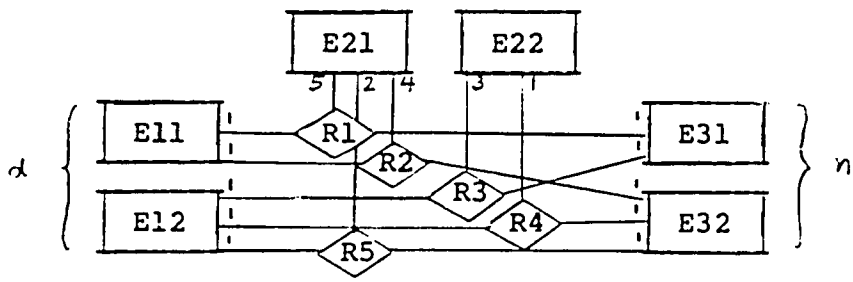


possible instantiation

Example 2 - find maximum cardinality of E2 in R (α 2).

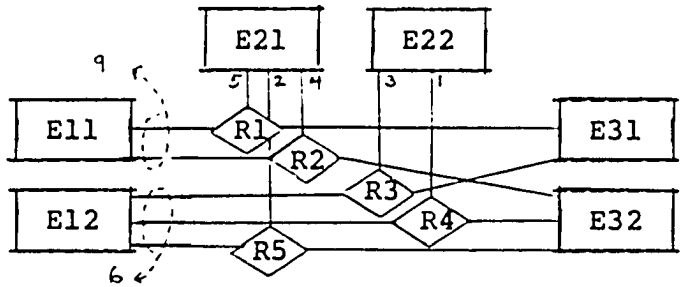


original R-graph (generic set)

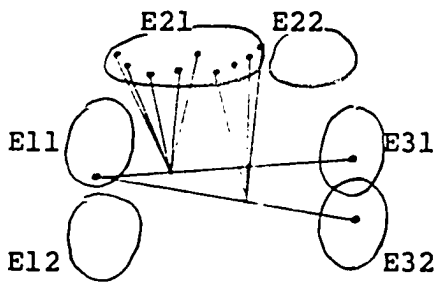


original R-graph (specialized sets)

case I - find α_2 from E11 and E12.

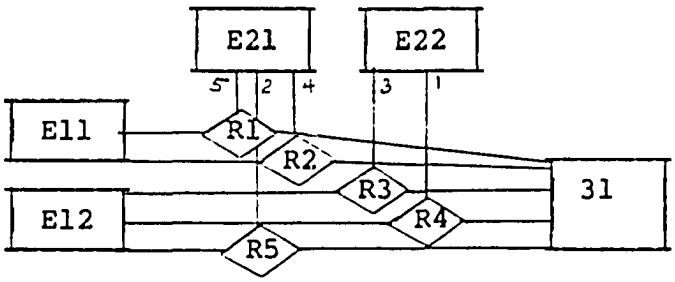


add up cardinalities for each E-node, $\alpha_{2_1} = 9$

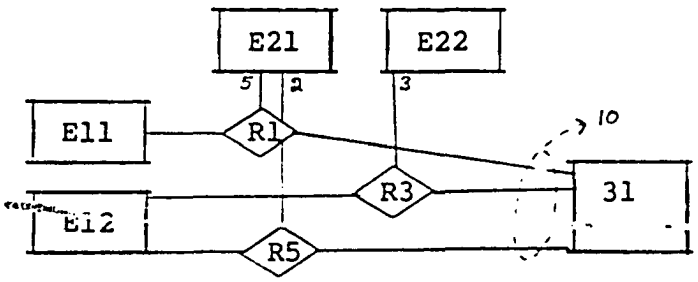


possible instantiation of case I

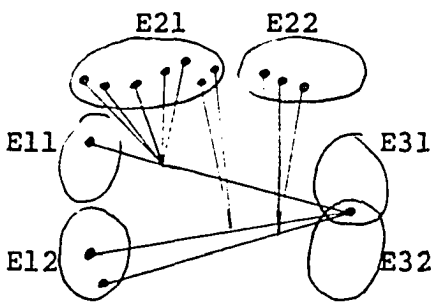
case II - find α_{2_2} from E31 and E32



combine E-nodes



combine R-nodes and add up cardinalities, $\alpha_2 = 10$

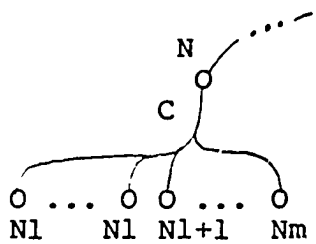


possible instantiation of case II

APPENDIX C

A PROOF FOR THE DETOUR ANALYSIS PROCEDURE

Let D be a detour of a road R in a graph D. Assuming that D is created from the nodes at level n of R. From the definition of detour, at least one path of D from the root node to a node at level n of R must be included in every instantiation of G. There are two aspects of the detour analysis procedure namely the mapping constraint invalidation and the null node identification. Assuming that the node N has the immediate successors N1,N2,...,Nm under the category C in a graph G.

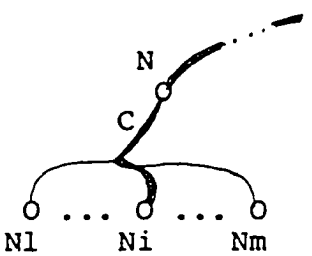


The mapping constraint of the categcry C under N in a graph G must be onto if the following condition is satisfied by all instantiations of G. If any instantiation of G which includes N must also include at least one of N1,...,Nm, then the mapping constraint of C must be onto.

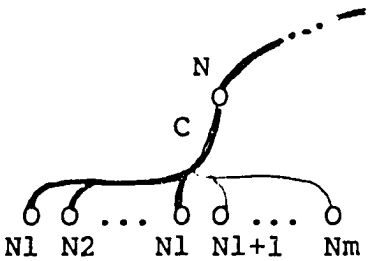
The nodes N1+1,...,Nm under the category C may be identified as null nodes if the following conditions are satisfied by all instantiations of G. If the mapping constraint of C is onto and N1,...,Nm are disjoint, then any instantiation of G which includes N must include exactly one

successor of N under C. In addition to the previous conditions, if any instantiations of G which include N must include at least one of Nl,...,Nl, then Nl+1,...,Nm may be identified as null nodes. This claim is not valid if the Nl,...,Nm are not disjoint. Since an instantiation of G which include some nodes from Nl,...,Nl may also include some nodes from Nl+1,...,Nm. Consequently, we can not claim that Nl+1,...,Nm are null nodes.

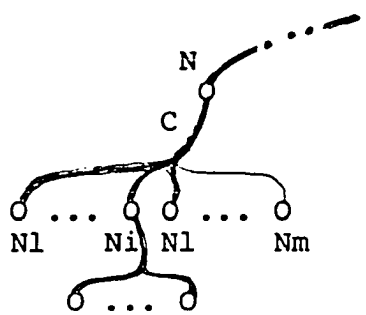
The detour analysis start from the root node of G and proceed downward. Let N be the current node during the analysis of D.



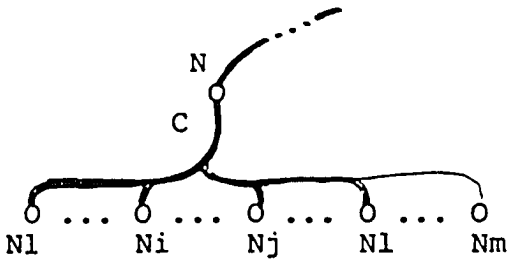
case I. N has only one category C and one immediate successor Ni under it in D. In all instantiations of G, any root originated paths of D which go through N must also go through Ni. This case satisfies the mapping constraint invalidation condition. Thus, the mapping constraint of C must be onto. If Nl,...,Nm are disjoint, then the conditions for null nodes identification are also satisfied. And the nodes Nj, with j=1,...,m and $j \neq i$, are null nodes. Since Ni is the only successor of N in D the analysis process may be repeat with Ni as the new current node.



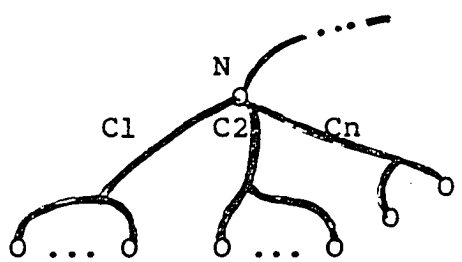
case II. N has the immediate successor N1,...,Nl under the category C in D. In all instantiations of G, if a root originated path of D goes through node N, then it must goes through one of the nodes N1,...,Nl. This condition is the same as mapping constraint invalidation condition, and the mapping constraint of C must be onto. If the nodes N1,...,Nm are disjoint, then the null node identification conditions are also satisfied. And the nodes Nl+1,...,Nm are null nodes.



If N1,...,Nm are disjoint, then in any instantiation of G any root originated path of D which goes through node N must go through exactly one of the nodes Ni (i=1,...,l) and no other sibling of Ni. So we can make the claim that if Ni is included in any instantiation of G, then at least one of its successor in D (if there is any) must be included also. Thus, the analysis procedure is repeated with each Ni (i=1,...,l) as the new current node in turn.



If Nl,...,Nm are not disjoint, then the detour analysis can not be extended to any successor of N. We can not claim that any instantiation of G which include the node Ni (i=1,...,1) must include some of its successors in D. Assuming that the node Ni and Nj ($i \neq j$ and $1 <= i,j <= 1$) are included in an instantiation of G. We may also assume that all root originated paths of D which go through N in this instantiation go through Nj only. Since Ni is not part of any path of D, none of its successor is required to be included. So, the analysis procedure can not be applied to any of the nodes Ni (i=1,...,1).



case III. N has no successor in D or N has more than one categories under it in D. Obviously if N has no successor at all in D, then the detour analysis must be terminated. If N has more than one category under it in D, then the process must also be terminated. Since there are more than one categories under N in D, an instantiation of G which includes N is not required to include any successors of N

under one particular category. As a result, the necessary conditions for mapping constraint invalidation and null node identification may not be satisfied by N. Hence, the detour analysis of D is terminated at N.