SISU rapport nr 11

Reverse modeling from Relational Schemata to Entity-Relationship Schemata

Katalin Kalman

SISU
Svenska Institutet för Systemutveckling
Box 1250, 164 28 KISTA

Reverse Modeling
from
Relational Database Schemata
to
Entity-Relationship Schemata

ISSN: 0282-9924

FÖRORD

I många organisationer finns idag ett stort antal databas- och informationssystemtillämpningar, som utvecklats kontinuerligt under en lång period. Ett problem med dessa system är att den exakta förståelsen av de data som finns i systemen ofta gått förlorad under årens lopp. Självfallet försvårar detta en effektiv användning av systemen, och det är därför viktigt att på olika sätt försöka bygga upp en bättre förståelse av de data som redan finns i en organisation. Ett sätt att göra detta är att beskriva data med hjälp av en konceptuell modell, som ligger på en högre abstraktionsnivå än traditionella datamodeller.

Det finns också andra fall då det är viktigt att kunna beskriva innehållet i en databas med ett konceptuellt schema, t ex om man skall förse en databas med ett gränssnitt i naturligt språk eller någon annan form av avancerat gränssnitt. Sådana gränssnitt kräver ofta mer information om de data som finns i databasen än vad som kan uttryckas i ett databasschema. Den ytterligare information som behövs kan då beskrivas i det konceptuella schemat.

Om man skall översätta ett stort databasschema till ett konceptuellt schema är det till stor nytta att ha tillgång till en metod som stöd i översättningsprocessen. En sådan metod kan inte reduceras till en algoritm som automatiskt översätter ett databasschema till ett konceptuellt schema. Eftersom konceptuella scheman kan innehålla mer information än databasscheman är det nödvändigt att en användare tillför ytterligare information vid översättningen. En viktig del av en metod blir därför att beskriva hur en användare på ett enkelt sätt skall kunna tillhandahålla den extra information som behövs. I SISU-rapporten *Reverse modeling from Relational Schemata to Entity-Relationship Schemata* av Katalin Kalman beskrivs tre olika metoder för översättning av relationsdatabasscheman till ER-scheman *(eng Entity-Relationship Schemata)*. Fördelar och nackdelar med de beskrivna metoderna diskuteras utförligt. Katalin Kalman har också skrivit ett program som implementerar en av metoderna. Denna rapport utgör även Katalin Kalmans examensuppsats för hennes *Master of Science*.

ACKNOWLEDGEMENTS

I would like to thank my advisor Janis Bubenko and I would especially like to thank Paul Johannesson and Benkt Wangler, my thesis advisors, who have generously contributed their time and expertise for this project and whose support I have found highly invaluable.

ABSTRACT

Both the database and the conceptual schema can be seen as structures representing relationships between objects in the real world. A conceptual schema, however, contains more semantic information, regarding these objects and relationships, than does the database schema. Generating a conceptual schema from a database schema is a process, known as reverse modeling, in which semantic information unavailable in the database schema is needed. This missing information must, in some way, be obtained by the modeling process. How much of this information is generated automatically and how much must be manually supplied by a user varies from method to method. It is argued that a high degree of automation is desirable in reverse modeling.

Two methods that generate a conceptual schema from a relational database schema are discussed. Their strengths and weaknesses are assessed and proposals for improvements are made. A third method, based on the strengths of the other two and building on suggestions for improvements for these methods, is then presented.

This new method makes use of a dictionary, containing synonyms of attribute names, for the modeling process. This dictionary becomes updated with each occasion a conceptual schema is created. This causes the system to improve with every use, thus, rendering it dynamic and enabling it to function as a learning system. Another advantage of the method is its high degree of automation resulting in a lesser need for user interaction.

The three methods, discussed, have been implemented as prototypes. They are fully functioning systems which take relational database schemata as input and produce graphical representations of conceptual schemata.

TABLE OF CONTENTS

1

INTRODUCTION

Databases are usually organized with the objective of making the data, contained therein, easily retrievable by various online and offline systems. These systems make use of the information in the database to produce reports as well as to display collections of data on a screen in response to online queries. They do not, however, produce a description of associations between concepts depicted by the data items. These associations constitute vital semantic information which often gets lost in the process of encoding information to create a database. This commonly results in a situation where the ones with access to the original information are unavailable and those inheriting these databases lack the understanding for using and maintaining them. This missing semantic information can be recaptured by extracting data objects from the database and showing interrelationships which may exist between these. A conceptual (semantic) model provides such an interpretation. This process of recapturing the missing semantic information is known as reverse modeling.

The task of representing a set of facts, in a database, in terms of a conceptual model can be carried out in a number of different ways. It can be done entirely manually, or with varying degrees of mechanization. However, regardless of the degree of mechanization chosen, it is necessary to have the proper tools and resources. This is an essential prerequisite for the successful completion of any task. The resource, in this case, is the set of facts; that is to say, a relational database. The tool is the method used for choosing the components (entities and relationships) of the conceptual model.

The facts, making up the database schema, must be organized in a manner which facilitates the formation of semantic relationships. If the original database is not structured in this way, it must be changed. This is a multi step process which is carried out by applying the tool to the resource as many times and in as many ways as is necessary to fine tune the database so that it eventually can be turned into a conceptual model.

Aside from shaping the resources, the tool is also used for choosing one semantic relationship when several potential ones exist. A number of different relationships can, sometimes, be extracted from the same set of facts describing a relational schema. Which particular structure is chosen, to express a situation, is dependent on the way the situation is interpreted. In such a case, it is the function of the tool to acquire the desired interpretation and extract the appropriate relationship from the set of facts in the database.

This tool which fine tunes the database, resolves ambiguous situations and makes other sorts of decisions, is, of course, the algorithm. In this paper, two such algorithms for creating a conceptual model out of a relational database are described, analyzed and criticized. These are, then, combined to yield an improved method. The first algorithm to be examined is one proposed by Navathe and Awong in [Navathe87]. The second of these has been proposed by Johannesson and Kalman in [Johannesson89]. Each of the three algorithms has been implemented in Prolog and fully tested.

2

CLARIFICATION OF CONCEPTS AND CONSTRUCTS

2.1 THE RELATIONAL DATABASE

A relational database, as described in [Korth86] and [Date86], is made up of relations. A relation is a table (that is to say, a two dimensional array) composed of attributes and values for the attributes. Each relation in the database has a unique name and each attribute in the relation has a unique name. An attribute or set of attributes which uniquely distinguishes an instance (a row containing the values of all the attributes of the relation) from any other instance in the relation is designated to be a key. A relation may contain several keys. One of the keys is designated as the primary key of the relation. The remaining keys are called candidate keys. An attribute or a set of attributes which is not a key is called a none key.

A relational database will be represented as shown in the two databases below. The name of the relation[1] will come first, outside of parentheses and will be written in capital letters. Attribute names will be contained in parentheses. Primary and candidate keys will always be underlined and none key attributes will not be underlined. Candidate keys will be written with hollowed letters.

Database:
HUNGRY_PERSON(<u>Person_Name</u>,<u>SS_num</u>)
WANTS(<u>Person_Name,Food_Name</u>)
FOOD(<u>Food_Name</u>)
SWEDISH_FOOD(<u>Food_Name</u>)
CHINESE_FOOD(<u>Food_Name</u>)

Database:
HOTEL(<u>Hotel_Name</u>,Addr)
ROOM(<u>Hotel_Name,Room#</u>)

2.2 THE EXTENDED ENTITY RELATIONSHIP MODEL

An EER model is a semantic model which is composed of entities, weak entities, relationships, and subclasses. An entity is an object in a particular environment. Entities are connected to each other by means of relationships. A subclass is an entity which fulfills a subclass role in a subclass/superclass relationship between two entity types. That is to say, the entity which is a subclass is a subset of the entity which is its superclass. A weak entity is an entity who's existence is dependent on another entity.

[1] For the sake of simplicity even a relation without instances (usually known as relation scheme) will be refered to as a relation in this paper. When a relation including instances is referenced , it will be explicitly stated that this is the case.

To illustrate the components of the extended ER model described above, graphical representations of the conceptual models corresponding to the database relations described in the previous section are presented in the figures below. In these representations, entities are denoted by rectangles, weak entities by a rectangle inside of another, relationships by diamonds, and subclasses by concave arcs placed on the connecting line between the subclass entity and its superclass. In the example to the left, *wants* is a relationship between entity types *hungry_person* and *food*, the entities *swedish_food* and *chinese_food* are subclasses to the entity type *food*. In the example to the right *room* is a weak entity dependent of the entity *hotel*.

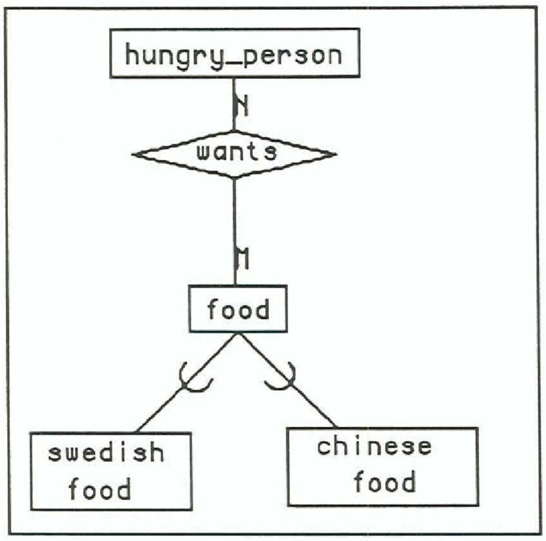


Figure 2.1 The EER model.

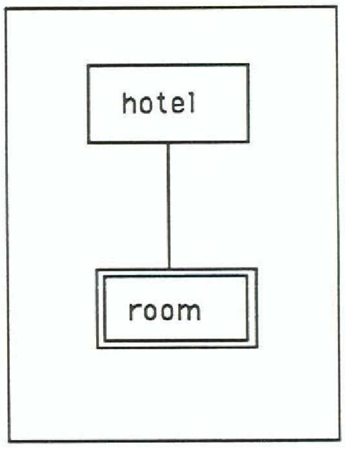


Figure 2.2 Weak entity.

2.3 REVERSE MODELING

In order to understand the term *reverse modeling*, or *reverse engineering* as it is sometimes called, it is necessary to first define the term *forward modeling/engineering*[2]. Forward modeling is a concept which generally refers to the process of encoding information in some way. It has been defined in [Chikofsky90] as the traditional process of moving from high-level abstractions and logical, implementation independent designs to the physical implementation of a system by following a sequence of going from requirements through to the implementation of the design. The implementation can be a computer program, a database, or any construct which reflects the original specification in some encoded form.

The aim of reverse modeling is to regain information lost in the encoding process of forward modeling. It is the process of going from an implementation, such as a database, to the original specifications which include an understanding of the intent and inner workings of the implemented system. Chikofsky perceives the goal of reverse modeling to be twofold. The first is to identify the system's components and their interrelationships. The second is to create representations of the system

[2] Since the words modeling and engineering are synonymous in this context, it is not necessary to write both, and , hence, only the word modeling will be used from here on in.

in another form or at a higher level of abstraction. In order to achieve this, there is often a need to add domain knowledge, external information, deduction or fuzzy reasoning to the subject system to identify higher level abstractions beyond those obtained directly by examining the system itself. Another definition, and one which, perhaps, is more tailored to the database area, can be found in [Bruce89]. Here, Bruce defines reverse modeling as the inference and documentation, to a specified level of detail, of models of data structures and business rules underlying one or more current or proposed data processing systems.

3

HISTORICAL BACKGROUND AND RELATED RESEARCH

3.1 GENERAL BACKGROUND

The most widely used approach for conceptual design is the ER (entity-relationship) model, originally proposed by Chen [Chen76]. The fundamental modelling constructs of the ER model are entities and relationships. Entities are the principal objects about which information is to be collected. Entities of the same kind are collected into entity types, e.g. Employee or Department. (In this paper entity will often be written instead of entity type and context will be relied upon to disambiguate the meaning; the same applies for relationship and relationship type.) Attributes are used to give the entities and relationships descriptive properties such as color and weight. Some attributes may be identifiers, i.e. they can be used to uniquely distinguish among the occurrences of an entity type. Relationships represent associations between entities.

To incorporate more semantics into the ER model several extensions of the model have been proposed, e.g. in [ElMasri85] and [Teorey86]. The most important of these is the addition of the subtype construct. A subtype relationship from one entity type to another expresses a subclass/superclass ISA relationship between the two entity types. An entity type is said to be a subclass of another entity type if every occurrence of the subclass is also an occurrence of the superclass. An example of a subtype relationship is CHINESE_FOOD ISA FOOD.

Some important contributions in the area of forward modeling in the field of databases include [Briand84] and [Shoval87]. Contributions to reverse engineering include [Briand87], [Davis84], [Nilsson84], [Dumpala83], [Davis87], [Navathe87], and [Johannesson89]. The two most complete methodologies proposed for translating from the relational model to an extended ER model seem to be [Navathe87] and [Johannesson89].

A brief discussion of each of the above mentioned papers, with the exception of [Navathe87] and [Johannesson89], follows. These two papers are discussed in great detail later in the paper and are, thus, not treated in this section.

3.2 RESEARCH IN FORWARD MODELING

The paper [Briand84] describes how to translate from an ER diagram into a relational schema. The entities and relationships of an ER diagram are translated to prolog facts. These facts are then translated into a Bachman diagram, made up of record types and set types. These are then, used to generate the relations of a relation schema.

This method is interesting for two reasons. One is that it takes an "expert system" approach, showing how one can explain the translation performed to a user. Second, it compares the ER model to semantic networks, pointing out how similar the two really are and showing how easily an ER diagram can be extracted from a semantic network.

[Shoval87] presents the system ADDS which is used to assist a database designer in designing a database schema. ADDS automatically creates a database schema out of a conceptual schema expressed as an information structure diagram of the Binary-Relationship Model. The database schema obtained consists of normalized record types.

3.3 RESEARCH IN REVERSE MODELING

The first paper to propose a mapping to an ER model is [Dumpala83]. Here, three methods are presented, to translate each of the three, network, hierarchical, and relational models into an ER model. The translation algorithm for the relational database is a seven step procedure which is based on the idea of first classifying the relations, then creating different data structures for the ER model based on the different classifications. The translation algorithm for the network model is a simple three step procedure where for each record type an entity is created, for each link a relationship is created and for each recursive link a relationship for the same entity is created. In the hierarchical model, all trees that can be connected are connected, then nodes are mapped to entities and edges to relationships. The paper [Navathe87] can be seen as an extension of this work in the case of the translation from the Relational Model. Due to this, the paper is important due to its historical interest.

In [Briand87], a method is presented for creating an extended ER diagram from a minimal cover of functional dependencies. A functional dependency graph representing the relations of the database, as well as a set of inclusion dependencies are used to create entities, relationships and a set of aggregations which are to make up the new ER schema. First, a reduced functional dependency graph is derived. The attribute components of this graph are, then, examined to determine if they participate in an inclusion dependency, if not they become primary keys of entities. Relationships are created out of nodes composed of several attributes. The information acquired from functional dependencies is used to produce M-M and N-1 mappings of relationships. Attributes are, then, assigned to entities and relationships according to functional dependencies.

The paper is fairly theoretical and not as easily accessible as [Navathe87], which is also due to the fact that it is not very well written. However, it seems to propose a robust and well working algorithm for creating an ER diagram.

In [Davis84] a method translating a conventional file system into an ER diagram is presented. First a modified functional data model which is the current physical model of the file system is extracted using the physical data units of the file. This intermediary physical model is then converted to a logical model by following a conversion algorithm to remove the physicalness of the data until a logical view is obtained. This new logical model contains the entities and relationships of the data and is the final ER conceptual data model.

Two methods of translation from a relational schema to an ER schema are discussed in [Davis87]. The first method described is one in which a relational database model is translated into an entity relationship data model in which the structure of the data along with the inherent behavior of the data are represented in the resulting model. The inherent behavior of the data is determined by inherent constraints such as the condition that each relationship be of a particular cardinality. The second method, and that presented in the paper, is one where an ER conceptual model is created which includes the explicit behavior of the data. By explicit behavior is meant those constraints that are tangential to the data model itself. Explicit behavior, for example, is the rule that the salary of an employee cannot exceed a certain amount, is comprised of constraints which are subject to change often. The algorithm for the second method chooses those constraints which have a high probability of modification to include in the resulting model. The two methods are then compared and contrasted. The second method is deemed more valuable.

In [Nilsson84], a method for translating a COBOL data structure into an ER schema is outlined. The translation is done by the COBOL DATA-SCANNER which takes a record description from the file section of the data division contained in a structure library (a file containing the record descriptions from all the programs of an application) and produces a SYSDOC conceptual schema. SYSDOC is a conceptual schema language which is similar to the traditional ER schema. Records in the record description are transformed into entity classes. Elementary items are transformed into data elements (attributes). Tables are transformed into entity classes which are then connected to the entity class, derived from the record, by a relationship. Other relationships must be specified by the user, as well as all relationship names. The resulting schema can, then, be expressed in terms of a conventional ER schema which is to be used to document and thereby maintain COBOL applications.

In [Winans90], Winans and Davis describe a method for translating an IMS database to an ER model. The built in tree structure of the hierarchical model of the IMS is used in the translation process. Input to the algorithm consists of IMS DataBase Definitions (DBDs) which are processed one at a time. Information about segments, fields, and indexes within the DBDs are collected and processed to build the RE-ERM (reverse engineered Entity-Relationship Model). This paper builds on the ideas presented in [Davis87] and likewise relies on the behavior of the data as well as on its structure to map to the RE-ERM.

4

AN ALGORITHM BASED ON ATTRIBUTE NAME SIMILARITY

4.1 INTRODUCTION

A method of reverse modeling, which maps a relational database to a conceptual schema is presented here. It is a method proposed by Navathe and Awong and further examined in [Kalman89]3. In [Navathe87] two algorithms are presented. One of these is a ten step procedure, based on similarity of attribute names in the relations of the database. Attributes which are the same are expected to have identical names throughout the database. It is by identifying like attributes of different relations, that connections between the relations of the database are established.

The data initially available to the mapping process consists of the relations of a database. The relations are always in the third normal form and are given without instances.

4.2 A TRANSFORMATION STRATEGY IN NINE STEPS

The algorithm, which maps a relational database to an ER schema, has been implemented, with minor alterations and a few omissions, in MAKEMODEL, a program written in Prolog. Input to the program is a relational schema, that is, relation and attribute names without instances represented as a set of Prolog facts. These facts are then processed by a nine step procedure which creates an extended Entity-Relationship semantic data model containing subclasses as well as mappings of functional dependencies. Generalization categories have been omitted here as the usage of the concept is restricted to Navathe and Awong's paper and has, thus, been deemed too narrow to be considered of enough value to be employed in this implementation.

The aim of this method is to atomize the process of transformation as much as possible. However, since more information is contained in the conceptual schema than in the original database, the process cannot be totally atomized. Whenever ambiguities, that cannot be resolved mechanically, arise, MAKEMODEL makes use of user interaction to obtain the clarification it needs to be able to proceed.

The strategy used by MAKEMODEL to transform Prolog facts depicting relations in a database to prolog facts representing a conceptual model can, thus, be described as a process which methodically fine tunes the database, using a mixture of automatic modification and user interaction. Although it is a nine step algorithm, it can also be divided into two distinct parts. Step1 constitutes the first part, the second part consists of the remaining eight steps. Step1 uses the candidate keys to restructure the relations for the purpose of making it possible, at a later stage, to find relationships between them. The candidate keys are in effect neutralized here as they no longer have any value to the transformation process after this step. All relations as well as their attributes are classified in this first step.

3 This chapter is an updated version of this paper.

A detailed account of the nine step algorithm used by MAKEMODEL is given below:

STEP1 - Three actions that perform different types of candidate key substitutions are carried out here:

action1 - The primary key of each relation in the relational database is read. After reading a primary key, the candidate keys of the rest of the relations are scanned. If a candidate key is found which is identical to the primary key under examination, and there exists a sub/superclass relationship between the two relations, that candidate key is changed to be the new primary key of the relation and the relation's former primary key is changed to be a new candidate key. To find out if such a subclass superclass relationship exists, a question is put to the user.

An example of how MAKEMODEL processes such a case:

Input: listing produced by MAKEMODEL:

```
account_holder
Primary Key: [acctnum]

person
Primary Key: [person_id]
Candidate Key: [acctnum]

company
Primary Key: [company_id]
Candidate Key: [acctnum]
```

Interaction between MAKEMODEL and the user:

```
In order to establish whether there exists a sub/super class
relationship between the relations shown below,
please answer the following question(s) by typing yes or no.

Is there a sub/super class relationship between:
company and account_holder?
y
Is there a sub/super class relationship between:
person and account_holder?
y
```

Result of processing after action1:

```
account_holder
Primary Key: [acctnum]

person
Primary Key: [acctnum]
Candidate Key: [person_id]
```

```
company
Primary Key:  [acctnum]
Candidate Key:  [company_id]
```

The purpose of candidate key substitutions performed by action1 is to discover some of the sub/superclass relationships between relations and to prepare these for further questioning later. In **step9** all sub/superclass relationships are processed.

action2- The primary key of each relation in the relational database is read. After reading a primary key, the candidate keys of the rest of the relations are scanned. If the primary key is found to contain a candidate key of another relation, the part of the primary key which is equivalent to one of the candidate keys is replaced by the primary key of the relation which contains that candidate key.

An example of how MAKEMODEL processes such a case:

Input: listing produced by MAKEMODEL:

```
car_sale
Primary Key:  [buyer_id,eng_serial_num,seller_id]
None Key Attributes:  [sale_date]

car
Primary Key:  [license_num]
Candidate Key:  [eng_serial_num]
None Key Attributes:  [make,model]
```

Result of processing after action2:

```
car_sale
Primary Key:  [buyer_id,license_num,seller_id]
None Key Attributes:  [sale_date]

car
Primary Key:  [license_num]
Candidate Key:  [eng_serial_num]
None Key Attributes:  [make,model]
```

The purpose of candidate key substitutions performed by action2 is to provide relations which have nothing linking them to each other by common elements in their primary keys with such links. The relation whose primary key was changed, now has a potential to become a relationship between two or more entities at some future time.

action3- The candidate key of each relation in the relational DB is read. After reading a candidate key, the primary keys of the rest of the relations are scanned. If the candidate key of the relation under examination is formed by concatenating two or more primary keys, that candidate key becomes the new primary key of the relation and the previous primary key is made into a new candidate key.

Input: listing produced by MAKEMODEL:

```
course_offering
Primary Key: [section_num]
Candidate Key: [course_num,instructor_num]

instructors
Primary Key: [instructor_num]

courses
Primary Key: [course_num]
```

Result of processing after action3:

```
course_offering
Primary Key: [course_num,instructor_num]
Candidate Key: [section_num]
```

The purpose of candidate key substitutions performed by action3 is to identify relations which will later become relationships between entities.

The next stage is that of classification. First, the relations are classified into primary type1 (PR1), primary type2 (PR2), secondary type1 (SR2), and secondary type2 (SR2):

PR1 -	Relations whose primary key does not contain a key of another relation. (Will map to entities).
PR2 -	Relations whose primary key is ID dependent on the key of a PR1 relation. To decide whether an ID dependence exists between two relations, first the primary key of a relation is examined to see if it contains the primary key of a PR1 relation. If this proves to be the case, the user is asked to confirm or deny an ID dependence between the two relations. (Will map to weak entities).
SR1 -	Relations whose primary key is fully formed by concatenating the primary keys of PR1 and/or PR2 relations. (Will map to relationships).
SR2 -	Relations whose primary key is partially formed by concatenating the primary keys of PR1 and/or PR2 relations. (Will map to relationships).

Next, the attributes of all the relations are classified into four categories: Key Attribute Primary (KAP), Key Attribute General (KAG), Foreign Key Attribute (FKA), and None Key Attribute (NKA).

KAP -	Attributes in the primary key of a secondary relation which also constitute a primary key of a primary relation. These are used later (in step6) when processing SR2 relations. These relations map to relationships, which connect entities whose entity identifiers are the same as the KAP of the SR2 relation to new entities created from KAGs.

KAG - The remaining attributes in the primary key of a secondary relation which are not KAPs. These give rise to new entities used in the transformation of SR2 relations.

FKA - None primary key attribute that is also a primary key of another relation. Used (in step7) to establish a relationship between the entity created by the relation which contains this attribute as an FKA and the relation which contains the attribute as its primary key.

NKA - None primary key attributes which are not FKAs.

STEP2 - When two or more primary relations have identical primary keys and there exists a secondary relation which contains this key, it must be determined which of these primary relations is related to the secondary relation. This is resolved by asking the user to identify the primary relation which is relevant.

Interaction between MAKEMODEL and the user:

```
The attribute ss_num in flies comes either from emp or pilot.
Does ss_num come from emp?
n
Does ss_num come from pilot?
y
```

Those primary relations identified as not being relevant to the secondary relation in question are gathered together in a list and saved to be used later (in steps 5 and 6).

STEP3 - Entity types are created here. For each PR1 relation, an entity type is asserted to the prolog database.

STEP4 - Weak entity types are created. For each PR2 relation, a weak entity type is asserted to the prolog database.

STEP5 - Relationship types are created. First all SR1 relations are read. Then those primary relations whose primary keys form the primary key of the SR1 relation are gathered together in a list. This list is then updated by deleting those relations (now entities) which have been marked in step2 as being irrelevant to the SR1. A relationship type containing the of list the entities which are connected to each other through this new relationship is created and asserted to the prolog database.

STEP6 - Relationship types are created from SR2 relations that contain both KAP and KAG type attributes. For each SR2 relation a list is produced containing the names of the entities whose identifiers are the same as the KAPs of the SR2 (the primary relations whose primary key is the same as the KAP) as well as the names of the KAG attributes of the SR2. Relations marked as irrelevant for the SR2 are removed as in step5.

STEP7 - Relationships derived from FKA type attributes are created. For each relation that contains an FKA type attribute, a relationship between the entity formed by that relation and the entities whose identifiers are the same as the FKA attributes contained in the relation is defined.

STEP8 - Binary relationships are labeled with their corresponding functional dependencies. Relationships, mapped from SR1 and SR2 relations, that connect exactly two entities are processed separately from other relationships. The user is asked to state the functional dependency which prevails between the two entities connected by a binary relationship.

Interaction between MAKEMODEL and the user:

```
In order to assess the functional dependency between |courses|
and |instructors| in relationship <course_offering>, please chose the
correct dependency listed below by typing the number 1, 2, 3, or 4.

1.  There is a functional dependency from |courses| to |instructors|.
2.  There is a functional dependency from |instructors| to |courses|.
3.  Both of the above functional dependencies exist.
4.  None of the above functional dependencies exist.
1
```

STEP9 - Subclasses are created. When entities with identical identifiers are detected, the user is asked to state whether there exists a sub/superclass relationship between these entities.

Interaction between MAKEMODEL and the user:

```
Please indicate whether there exists a sub/super class
relationship between the following entities.

E1->E2 denotes that E1 is a subclass of E2.

emp->person
y
emp->pilot
n
pilot->emp
n
person->pilot
n
pilot->person
y
```

For each subclass relation identified, `subclass(E1,E2)` is asserted. `E1` represents an entity which is a subclass of the entity represented by `E2`.

4.3 MAKEMODEL: AN IMPLEMENTATION OF THE ALGORITHM

Implementing a method renders it testable. This facilitates the process of evaluation, which is done for the purpose of determining the validity of a method. It is, first, by testing with different examples that it becomes possible to infer something about the accuracy of a method.

MAKEMODEL, an implementation of the algorithm presented in [Navathe87], was constructed for the purpose of examining that method. A program description is presented below.

Program Description:
To activate MAKEMODEL, the user types the word start followed by a period. MAKEMODEL, then, begins its processing by requesting the user to select a relational database. The user may, choose to use an existing database, create his/her own database, or supplement an existing database with new relations.

After establishing an input database to MAKEMODEL, the user is given an opportunity to view the database. If this option is chosen, then every relation in the database gets displayed. For each relation, its name is printed out followed by the attributes which constitute its primary key, the attributes which constitute any and all candidate keys the relation may embody, as well as any existing none key attributes.

The nine steps making up the algorithm are executed next. This is done by the module 'process RDB to CM' which is the principal component of MAKEMODEL as it is where the mapping from a relational database to a conceptual model takes place. All questions pertinent to the processing of the particular database, given as input to the current run, are asked here and the generated structures corresponding to the conceptual model are asserted to the Prolog data base.

Finally, the user is given a chance to view the conceptual model created by MAKEMODEL. This step may be bypassed, in which case, a concluding remark is printed out and MAKEMODEL is exited. If the user wishes to see the model, then information about each entity and relationship produced is printed out and graphical representations of relationships between the entities are drawn out on the screen.

4.4 TESTING THE ALGORITHM

Prior to reaching the level of competence necessary to criticize any method, it is important to gain a reasonably good understanding of the method in question. One way to accomplish this is by using pragmatic means. The behavior of the method could be observed under various conditions. This would make it possible for different aspects of the method to be seen, thereby enabling one to acquire a better understanding of the method as a whole.
In this case, the algorithm has been closely scrutinized by testing MAKEMODEL quite thoroughly. A vast number of examples were used as input to MAKEMODEL and the results carefully noted and recorded.

To appreciate the results obtained from using test examples, it can be helpful to keep in mind the different types of relationships which can arise and how MAKEMODEL creates these. A brief recapitulation follows.

Four different types of relationships can prevail between entities. The first is a relationship between a subclass entity and its superclass. The second is a relationship between an entity and a weak entity. The third is a relationship between two or more entities in which the relationship has a name, a relationship identifier and none key attributes. Finally, there is the relationship between two entities where the relationship has neither a name nor identifier.

The possibility of a relationship between a subclass entity and its superclass is recognized whenever there exist two or more relations with the same primary key. This situation is first isolated in step1.action1, through a user question, whenever a candidate key of a relation matches the primary key of another relation. The other two candidate key substitutions can also give rise to this situation. The place where sub/superclass relationships are actually created is step9. Weak entities are recognized via user interaction in step1. They become classified as PR2 (primary relations) and the relationship is finally created in step4. Ordinary relationships between two or more entities are recognized by the fact that their primary keys contain primary keys of other relations. These relations get labeled as SR1 or SR2 (secondary relations) in step1 and are processed in different ways, depending on various conditions, in steps 5,6 and 8. The last type of relationship is the FKA. These are created from relations containing attributes which are also primary keys of other relations. These attributes are labeled as FKA in step1 and are then set up as relationships in step7.

It is not difficult to see that the creation of relationships is a rather elaborate process. The manner in which a relation is turned into an entity or a relationship is rather opaque and it is, thus, not intuitively clear that the model generated is the correct one.

The examples used to test the algorithm, were, for the most part, arbitrary databases selected from various domains. Many were also somewhat large and rather intricate. The reason for choosing large examples, is that the shortcomings of the algorithm are, more likely, found by using examples where more complex situations are depicted.

4.5 CRITIQUE OF THE ALGORITHM

In order for the algorithm to be able to function at all, an important expectation about the input database must be met. The database must be set up in such a way that identical attributes have the same names. Databases are normally not set up in this way. Identical attributes may have identical names but the names could, just as well, be somewhat varied or abbreviated in some of the relations. This means that the database must be changed by hand in order to fulfill the prerequisite required by the method. This presents a considerable weakness in the algorithm.

Aside from this weakness caused by the necessity of like attributes having like names, a number of other problems have been found. Ten important and basic problems inherent in the algorithm and made apparent by testing MAKEMODEL are presented below. Each problem is described, the correct model which should have resulted is shown, and a solution is proposed.

(1) - Multi level ID dependence

Problem description:
The situation in which an entity is ID dependent on another entity which in turn is ID dependent on a third entity cannot be produced. The reason for this is that according to the algorithm an entity can only be ID dependent on a PR1 relation.

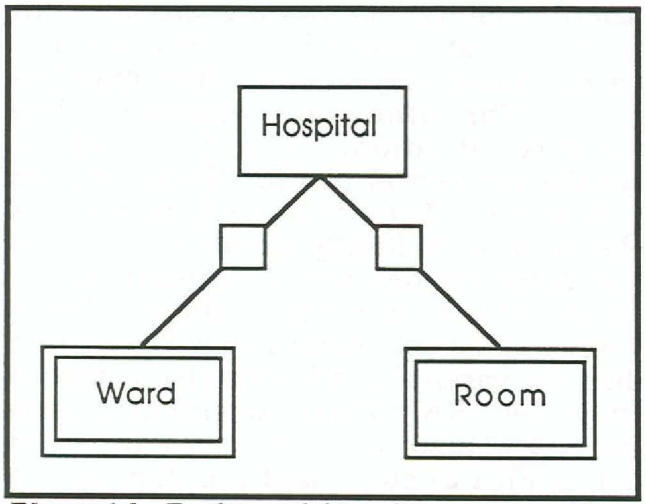


Figure 4.1. Faulty model involving weak entities.

In this example the entity Room is ID dependent on the entity Ward which in turn is ID dependent on the entity Hospital, but this cannot be represented by MAKEMODEL. Instead, the two relations Ward and Room become weak entities ID dependent on Hospital, an entity created out of a PR1 relation.

The resulting faulty model is shown in Figure 4.1 to the left.

Proposed solution:
This can be solved by letting a relation be ID dependent on none PR1 relations as well as on PR1 relations.

Figure 4.2 below shows the correct conceptual schema for the example discussed above.

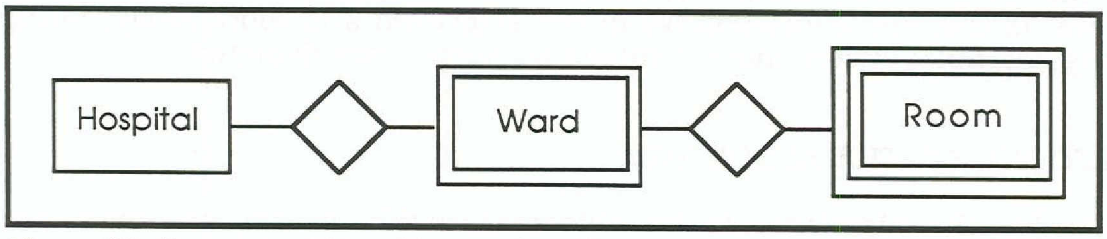
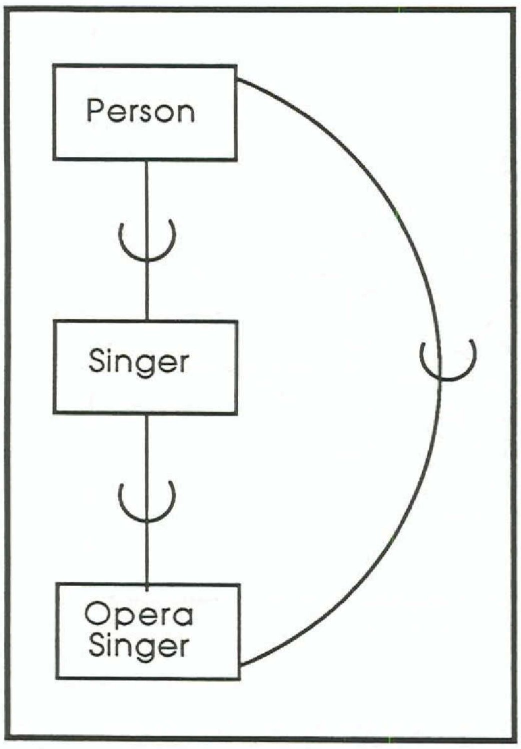


Figure 4.2 Correct model involving weak entities.

(2) -Sub/super class relationships and the indirect links between these.

Problem description:
When two or more levels of sub/super class relationships exist, the indirect links should only be represented implicitly in the conceptual model. i.e. if there were a subclass relationship from an entity C to an entity B and a subclass relationship from entity B to an entity A, then the subclass relationship from C to A should not be depicted in the conceptual model. These indirect links do, however, become allocated. Since sub/super class relationships are established through user interaction, the question as to the existence of a connection between A and C would be

answered in the affirmative and a direct link between A and C would then be set up in the conceptual schema.



The link between Opera singer and Person should not be represented explicitly in the conceptual model.

Figure 4.3 to the left shows the faulty model created by MAKEMODEL.

Figure 4.4 to the right shows the correct conceptual schema.

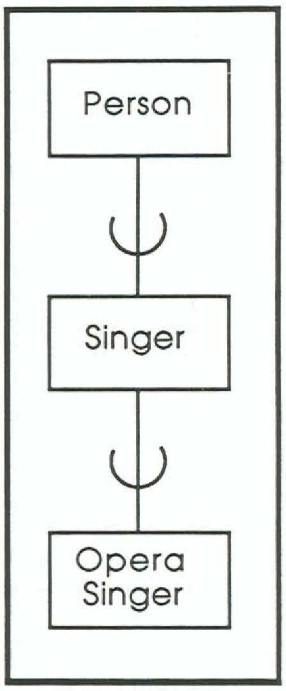


Figure 4.4

Figure 4.3 Indirect link.

Figure 4.4 Indirect link not represented.

Proposed solution:
This problem can be solved by deleting indirect links after all super/subclass links for a particular primary key have been specified by the user.

(3) -<u>FKA relationship between an entity and one participating in a sub/superclass relationship.</u>

Problem description:
Since a subclass superclass relationship between entities is mapped from relations which have identical primary keys, a relationship between one of these and another entity (created via FKA) will automatically establish a relationship between the other entity and all entities which participate in the subclass superclass relationship with that entity. This can result in the creation of superfluous as well as even incorrect relationships. This problem is taken up and solved in the algorithm for the particular case of relationships created out of SRs (secondary relations). The solution prescribed is to ask the user to identify the correct entity or entities that partake(s) in a relationship whenever there are several candidates (primary relations with identical primary keys).

In the example shown in Figure 4.5 the entity Course should only be related to the entities Teacher and Student.

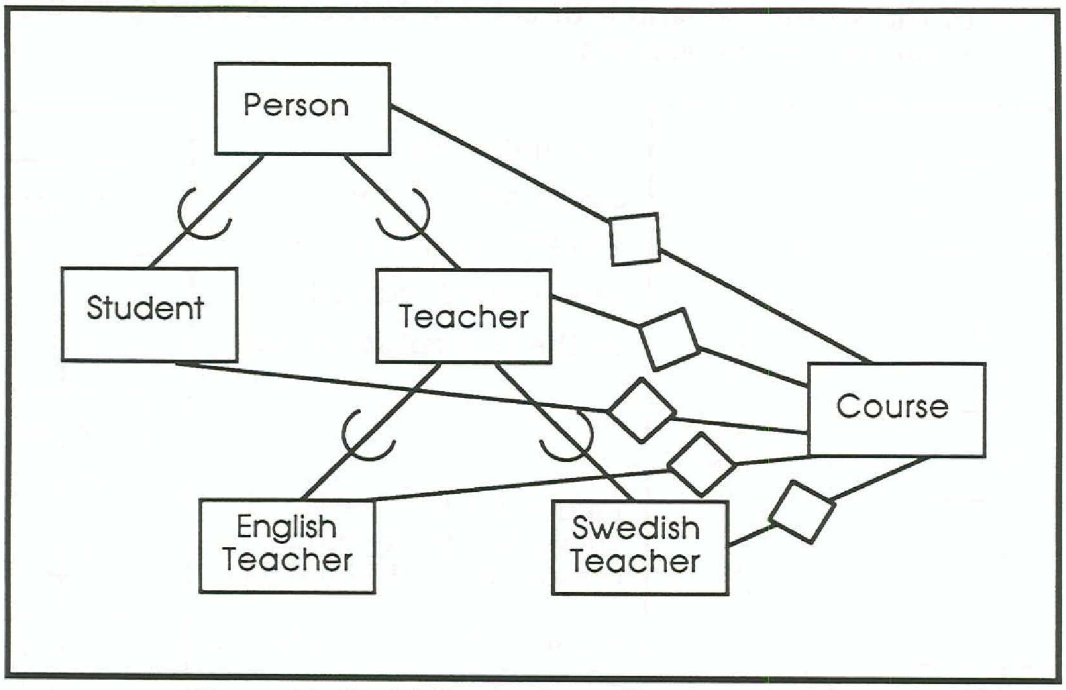


Figure 4.5 Establishment of superfluous relationships.

Proposed solution:
This can be resolved by asking the user to specify which of the super and subclass entities are related to the other entity through an FKA attribute. The questions will be almost identical to the ones asked in step2 whenever there exist SRs that have attributes in their primary keys which match the primary keys of several PRs.

Figure 4.6 below shows the correct conceptual schema.

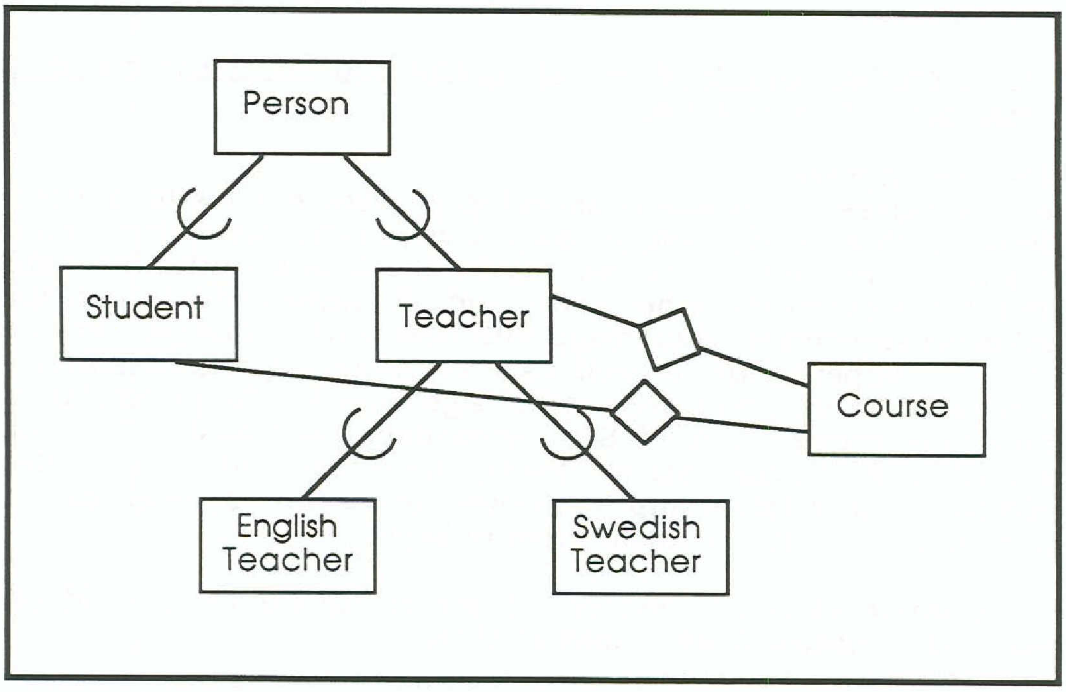


Figure 4.6 The correct schema without superfluous relationships.

(4) - Creation of several new entities from one primary relation.

Problem description:
Sometimes, a primary relation (PR1 or PR2) should give rise to two or more entities. Navathe's algorithm cannot handle such cases.

Example of such a case:
COUNTRY(Name,Currency).
INFLATION_RATE(Currency, Year,Rate).

This should produce the model shown in Figure 4.7 below:

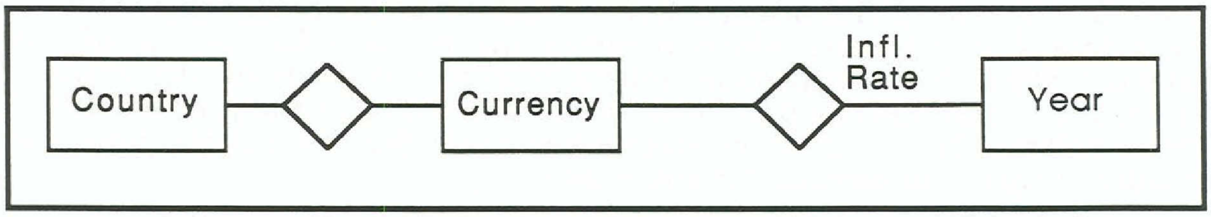


Figure 4.7 Creation of the extra entity Currency.

Instead MAKEMODEL produces:

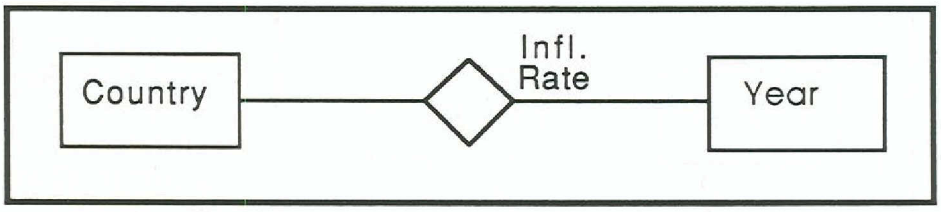


Figure 4.8 Faulty model without the entity Currency.

Proposed solution:
An enhancement to the algorithm, in order to handle such cases, could be made in the following way. For each attribute in all candidate keys of a relation, ask the user if the attribute should be mapped to an entity. If the user answers yes, create a relation out of the attribute, and let its primary key be composed of one attribute whose name is the same as the name of this new relation. Take the attribute out of the candidate key of the first relation and add it to the none key attributes of the same relation. This should be carried out before any candidate key substitutions are made.

(5) - Candidate key substitutions to create sub/superclass relationships.

Problem description:
The candidate key substitution, carried out in step1.action1, which prepares relations for becoming entities that are to be subclasses of other entities, cannot handle the situation in which there is more than one superclass to a subclass. When a relation X has several candidate keys which match the primary keys of several other relations, say Y and Z, and there is a subclass/superclass relationship between the relations X and Y as well as between X and Z, the algorithm is unable to establish both of these connections.

Example:
PIANO(P#,F#,I#).
INSTRUMENT(I#,Weight).
FURNITURE(F#,Length).

After establishing a sub/super class relationship from PIANO to INSTRUMENT, the candidate key I# becomes the primary key of PIANO and P# becomes a candidate key. At this point there is no longer a possibility of establishing a connection between piano and furniture.

Proposed solution:
Instead of replacing the primary key of a subclass relation with one of its candidate keys (the one which matches the primary key of one of the relations which has a superclass relationship to this relation), the primary key of the superclass could, in some way, be included in the primary key of the subclass. The matching candidate key could then be deleted from the relation which is to become the subclass entity.

After the proposed candidate key substitution, the PIANO relation from the example above would be PIANO(P#,I#,F#).

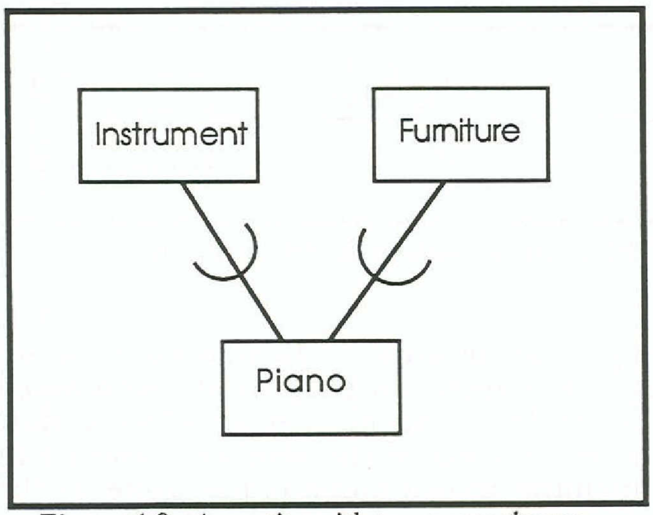


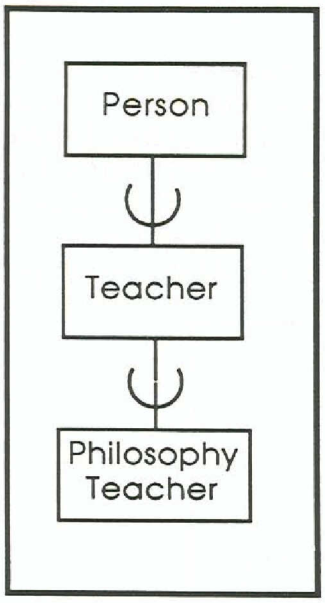
Figure 4.9 An entity with two superclasses.

After the proposed candidate key substitution, the piano relation from the example above would be: PIANO(P#,I#,F#).

The correct model that would result is shown in Figure 4.9 to the left.

The following is an example involving two levels of sub/superclass relationships.

PERSON(SS_num)
TEACHER(Teacher_num,SS_num)
PHIL_TEACHER(P_num,Teacher_num)



After the proposed candidate key substitution, the relations from the example above would look like this:

PERSON(<u>SS_num</u>)

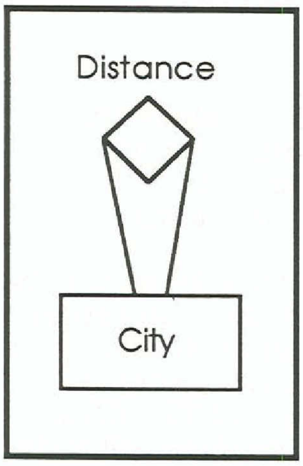
TEACHER(<u>Teacher_num,SS_num</u>)

PHILOSOPHY_TEACHER<u>(P_num,Teacher_num)</u>

The resulting conceptual schema is shown in Figure 4.10 to the left.

Figure 4.10 Two levels of sub/superclasses.

(6) - <u>Relationship from and to the same entity.</u>
Problem description:
The algorithm is not equipped to express the situation where an entity has a relationship to itself. An example of this is a relationship distance which connects two cities together. This should produce the conceptual model shown in Figure 4.11.



In the example taken up in this section the relationship distance connects two cities together. The figure to the left illlustrates the correct representation of this situation.

Figure 4.11 Relationship between one entity.

Proposed solution:
One solution to this problem is to supply the algorithm with two relations on input, one, the relation representing the entity, the other a relation whose primary key contains the primary key of the first relation two times. For the example above the relations supplied would be: **city(city)**, **distance(city,city)**. A relationship (distance) would then be created between two entities (city and city), but since only one such entity exists,

the relationship actually connects the entity (city) with itself. This is an unnatural way of solving the problem.

Another way to represent this scenario is to have a relationship connecting two entities that have a subclass relationship to an entity. Unfortunately, the relation which is to become the relationship between these two entities will still have to contain two identical attributes in its primary key. The reason for this is that relations that get mapped to subclass entities have the same primary keys as the relation which is to become their superclass entity. Fig. 4.12 shows the resulting conceptual model.

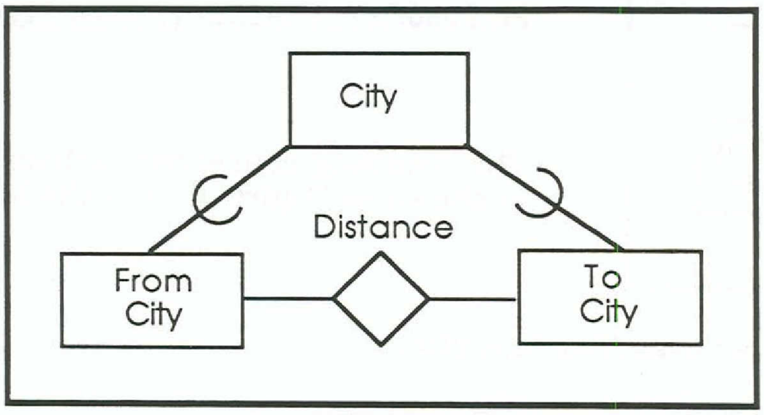
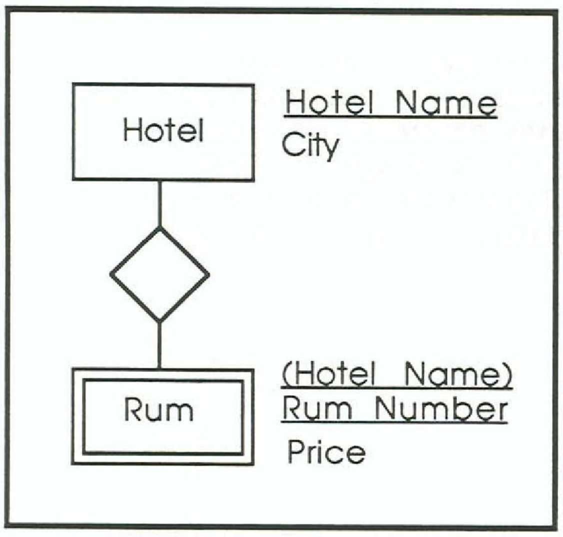


Figure 4.12 Relationship between two subclasses of the same entity.

(7) - <u>Representation of inheritance of attributes by weak entities.</u>

Problem description:
Since a weak entity is ID dependent on another entity, it should inherit the attributes of the entity on which it is ID dependent. This seems to have been overlooked in the algorithm.



Hotel_Name which is the identifying attribute of the entity Hotel, is inherited by the weak entity Rum.
This inheritence is implicitly contained in the conceptual representaion of the ID dependence relationship produced by MAKEMODEL and should not be explicitely represented as the identifier of the weak entity Rum.

Figure 4.13 Attribute inheritance.

Proposed solution:
In a similar fashion as inherited attributes become deleted from entities that have subclass relationships to other entities, inherited attributes of weak entities should also be deleted.

(8) - Establishment of faulty connections between relations.

Problem description:
The candidate key substitution, carried out in step1.action2, which establishes a connection between a relation whose candidate key is included in the primary key of another relation with that relation, can create faulty relationships between entities by establishing unnecessary connections between relations.
The following example illustrates such a case:
CITY(Cityname).
CITYPOPULATION(Cityname,Year),
COUNTRY(Countryname,Cityname).

After candidate key substitutions have been made, the relation citypopulation is altered to be CITYPOPULATION(Countryname,Year).

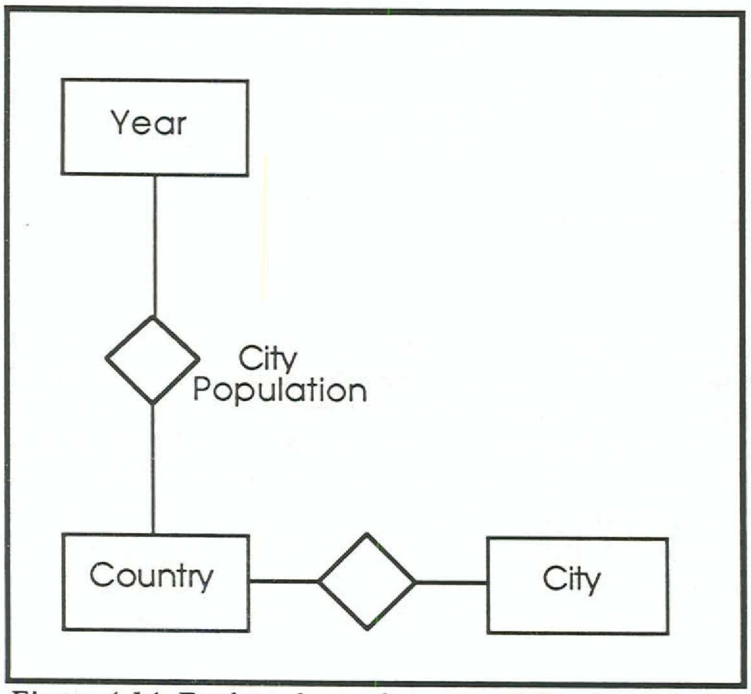


Figure 4.14 *Faulty schema due to candidate key substitutions.*

After the candidate key substitution shown above has been carried out, a connection between the relation country and the relation citypopulation is established. Citypopulation becomes an SR2 relation giving rise to a relationship between the entity country (created from the relation country) and year (an entity created out of the attribute year in citypopulation). The entities city and country become linked through the FKA cityname in the relation country.

This results in the faulty model shown in Figure 4.14 to the left.

Without the faulty candidate key substitution a different conceptual model would be produced as shown below.

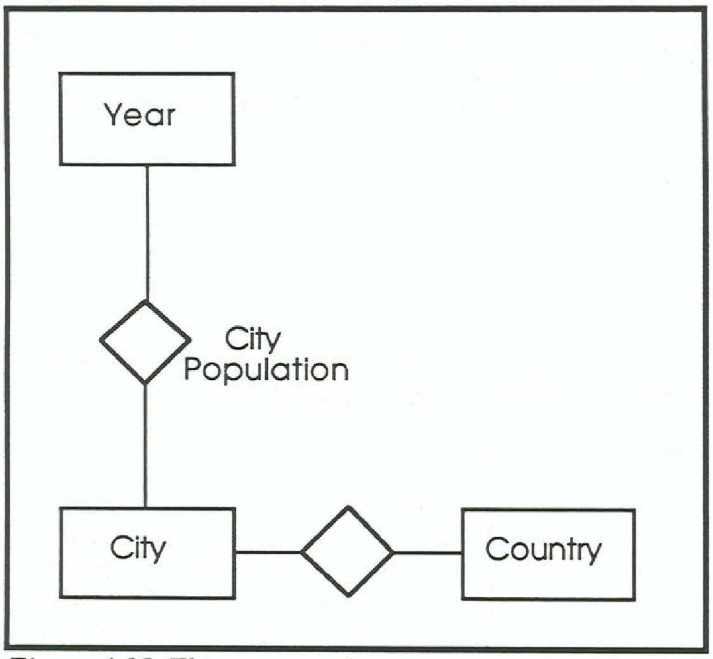


Figure 4.15 The correct schema.

If the attribute cityname in country had been a none key attribute instead of a candidate key, a connection between country and citypopulation would never have been established. The existing connection between city and citypopulation would have been exploited, giving rise to the relationship citypopulation connecting the entity city with the entity year. City and country would still have been related through the FKA cityname.

This would have resulted in the correct model shown in Figure 4.15 to the left.

Proposed solution:
A more discriminate use of the candidate key substitution carried out in step1.action1 would eliminate some of the unnecessary connections which may be created between relations. Before replacing an attribute in the primary key of a relation by a primary key of another relation, the primary keys of the remaining relations in the database should be examined. If a primary key of one of these is found to match the attribute which is under consideration for replacement, it is an indication that there already exists a connection between the relation for which there is an attempt to define a connection and another relation and the replacement should thus not be carried out.

(9) - <u>Incapability of connection an FKA to an SR relationship.</u>

Problem description:
An FKA relationship cannot be established from an entity to a relationship created out of an SR2 relation. The reason for this is that since an FKA relationship is established by finding a primary key of a relation which matches a none key attribute of another relation, the primary key can only contain one attribute. This rules out SR relations.

Proposed solution:
For each KAG attribute in an SR2 relation (that is to say, attributes, found in the primary key of relationships, which do not constitute the primary keys of PR1 relations), a database search can be conducted to look for PR1 relations whose NKAs contain these KAGs. This establishes a type of an FKA connection between an SR2 and a PR1 relation. These FKA

relationships would, thus, serve to identify the entity to which the KAG attribute refers and to connect that entity to the appropriate relationship.

Figure 4.16 illustrates an example of this situation as handled by the proposed solution described above. In this example the entity child had an FKA connection to the relationship marriage. This can be seen in the database relations shown below.

MARRIAGE(<u>Wife_ID,Husband_ID,Family_ID</u>)
WIFE(<u>Wife_ID</u>)
HUSBAND(<u>Husband_ID</u>)
CHILD(<u>Child_ID</u>,Family_id)

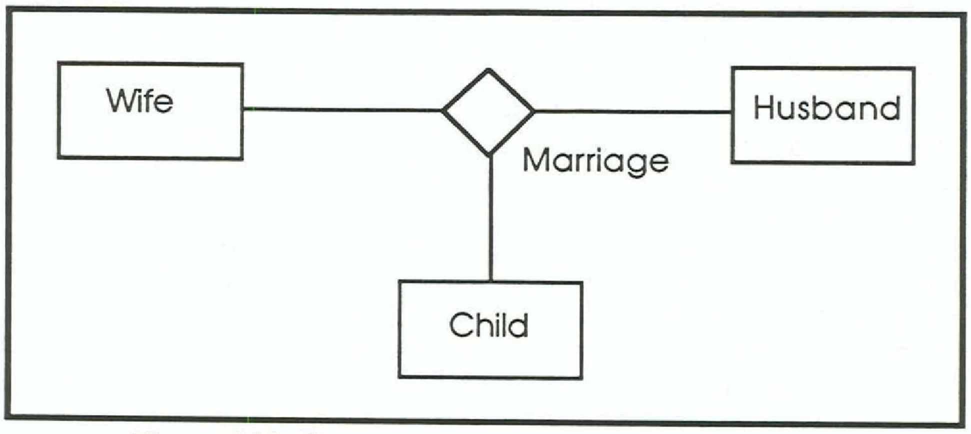


Figure 4.16 Connecting Child to Marriage via an FKA.

(10) - <u>Inappropriate entity names created from attributes.</u>

Problem description:
Entities created from the unaccounted for (KAG) attributes in the primary key of an SR2 relation can get names which are unsuitable for an entity.

example:
The following database:
SURGEON(<u>SS_num</u>).
WORKS_IN(<u>SS_num,Hosp_name</u>).

Would give rise to the schema shown in Figure 4.17.

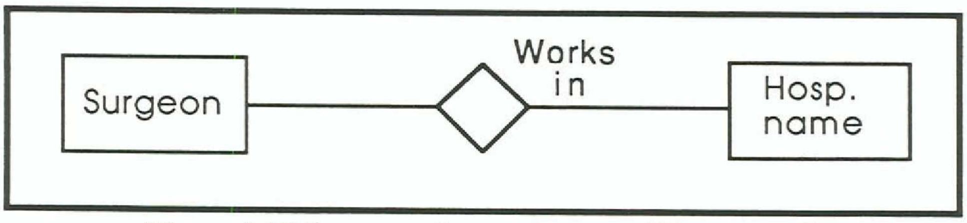


Figure 4.17 Inappropriate entity name "Hosp name".

Proposed solution:
Consult the user. Whenever a new entity is to be created from an attribute found in the key of an SR2 relation, ask the user to specify the name the new entity.

4.6 CONCLUDING EVALUATION

There appear to exist several serious flaws in the method presented by Navathe and Awong. Many, are fundamental errors which seem to have been overlooked while constructing the method.

Most of these problems can be solved with varying degrees of difficulty. To solve the problem of having to have identical names for identical attributes, a radical change must be made to the algorithm. Solutions for each of the other ten problems outlined in the previous section have been recommended and many of these have been implemented in a revised version of the program entitled 'FIXED MAKEMODEL'. The fact that the problems can be solved, does not, however, trivialize their severity.

Most of the recommended solutions require a considerable amount of user interaction. The result is that, to a large degree, it is the user who ends up solving the problem. Such a solution is in direct opposition to the nature and purpose of the method in question, namely that of mapping a database to a semantic model by the most automated means possible, which is to say, by keeping the amount of user interaction to a minimal level.

Constructing a semantic data model from an ordinary database, often requires the presence of a certain amount of semantic information which is not, ordinarily, contained in the database. An algorithm which is to carry out this mapping process must obtain this information in some way. The information can either be supplied by a user or derived by other means. Generally, some combination of both, user input and mechanical generation, is used. The ratio of these two means, called for by the method, determines, to a large part, the quality of the system produced. This ratio is (as argued above) unsatisfactory in the method under examination.

5

AN ALGORITHM BASED ON INCLUSION DEPENDENCY

5.1 INTRODUCTION

The method presented here, and originally described in [Johannesson89][4], makes use of inclusion dependencies to obtain the information necessary to map relations in a relational database to a conceptual model. Inclusion dependencies express connections between the relations in a database. These connections are, then, used by the mapping process to give rise to various types of relationships between entities in a conceptual model.

The data initially available to the mapping process consists of the relations of a database as well as of a number of statements representing inclusion dependencies. An inclusion dependency is made up of two parts, each of which contains a reference to a relation as well as to a key or non key of the relation.

5.2 A TRANSFORMATION STRATEGY IN FOUR STEPS

The algorithm using inclusion dependencies to map a relational database to an ER schema has been implemented in MAKEMODEL2, a program written in Prolog. Input to the program is a relational schema as well as a set of inclusion dependencies. Inclusion dependencies are given in the following way: A.a << B.b, where A and B are relations, a is an attribute or a list of attributes of A and b is an attribute or a list of attributes of B. This inclusion dependency states that the set of values appearing in A.a must be a subset of the set of values appearing in B.b. The relations of the schema are always in the third normal form represented as Prolog facts. These facts are, then, processed by a four step procedure which creates an extended Entity-Relationship semantic data model.

A detailed account of the four step procedure used by MAKEMODEL2 is given below:

STEP1 - The relations of the database schema are classified into three types: primary, secondary and ternary relations.

PR A primary relation is a relation with the property that no true subset of its primary key occurs on the left hand side of an inclusion dependency. The entire key may, however, occur on the left hand side of an inclusion dependency. (Will map to an entity).

SR A secondary relation is a relation whose primary key is equal to the concatenation of the left hand sides of at least two inclusion dependencies. (Will map to a relationship).

TR A ternary relation is a relation such that some attribute of its primary key occurs on the left hand side of an inclusion dependency and some other attribute of the primary key does not occur on the left hand side of any inclusion dependency. (Can map to an entity or a relationship).

[4]This chapter is an updated version of this paper.

Example 1.1:

```
PERSON(SS#, Address)
EMPLOYEE(Emp#, Salary)
DEPARTMENT(Dept#, Floor)
WORKS_FOR(Emp#, Dept#, Start_date)
EMPLOYEE.Emp# << PERSON.SS#
WORKS_FOR.Emp# << EMPLOYEE.Emp#
WORKS_FOR.Dept# << DEPARTMENT.Dept#
```

In the above example PERSON, EMPLOYEE and DEPARTMENT are primary relations, while WORKS_FOR is a secondary relation.

Example 1.2:

```
HOTEL(Hotel#, Address)
ROOM(Hotel#, Room#)
ROOM.Hotel# << HOTEL.Hotel#
```

In the above example HOTEL is a primary relation and ROOM is a ternary relation.

Example 1.3:

```
COUNTRY(Name, Population)
MEMBERSHIP(Country, Organization)
MEMBERSHIP.Country << COUNTRY.Name
```

In the above example COUNTRY is a primary relation and MEMBERSHIP is a ternary relation.

STEP2 - Entities and relationships corresponding to the relations of the database are created.

Every primary relation will give rise to an entity. A secondary relation can give rise to a relationship. A ternary relation can give rise either to an entity or a relationship. The user is asked which modelling construct he prefers. Example 1.2 shows a case where a TR relationship gives rise to an entity, whereas in example 1.3 it gives rise to a relationship. All binary relationships introduced in this step will be of type N-N. Those relationships other than binary are not type identified.

STEP3 - All inclusion dependencies are treated in this step. An inclusion dependency of a relational database schema can influence the corresponding conceptual schema in essentially four different ways.

The first possibility is that of a subtype relation between two entities. The second possibility is that the inclusion dependency indicates a relationship between entities. In the third case the inclusion dependency gives rise to a new entity and a subtype relation. In the fourth case a new entity is to be introduced and there is to be a relationship involving this entity. A close interaction between keys and inclusion dependencies will determine which modelling constructs will be used in the conceptual schema to represent an inclusion dependency.

Case 1: Introduction of a subtype relation.

Let I = A.a << B.b be an inclusion dependency. If A corresponds to an entity, A.a is a key, B corresponds to an entity and B.b is a key then it is possible that there is to be a subtype relation from the entity corresponding to A to the entity corresponding to B. The attributes of B.b will give rise to identifying attributes of the entity corresponding to B.

Example 3.1:
PERSON(SS#, Address)
EMPLOYEE(SS#, Salary)
EMPLOYEE.SS# << PERSON.SS#

In this example the inclusion dependency will give rise to a subtype relation from EMPLOYEE to PERSON. The attribute SS# in PERSON will become the identifier of the entity Person (not shown on the graphic representation of the conceptual schema).

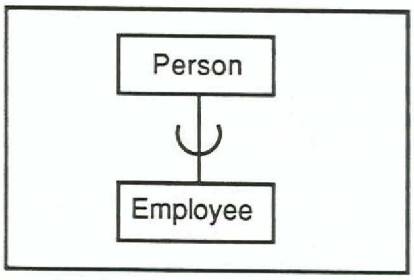


Figure 5.1 Result of EMPLOYEE.SS# << PERSON.SS#.

Example 3.2:
PERSON(SS#, Name, Address)
EMPLOYEE(Name, Salary)
EMPLOYEE.Name << PERSON.Name

In this example the inclusion dependency will give rise to a subtype relation from Employee to Person.

Case 2: Introduction of relationships between existing entities. This case can be divided into the following two sub cases:

Case 2a: Let I = A.a << B.b be an inclusion dependency. If A corresponds to a relationship, A.a is a non key contained in the primary key of A, B corresponds to an entity and B.b is a key then it is possible that the entity corresponding to B is an entity participating in the relationship corresponding to A. The attributes B.b will become identifying attributes of the entity corresponding to B.

Example 3.3:
EMPLOYEE(Emp#, Salary)
WORKS_IN(Emp#, Dept#)
WORKS_IN.Emp# << EMPLOYEE.Emp#

In this example the entity Employee will become an entity participating in the relationship.

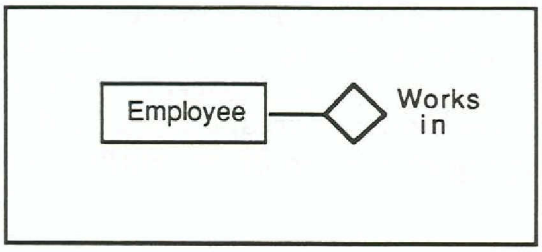


Figure 5.2 Result of WORKS_IN.Emp# << EMPLOYEE.Emp#.

Case 2b: Let I = A.a << B.b be an inclusion dependency. If A corresponds to an entity, A.a is a candidate key or a non key, B corresponds to an entity and B.b is a key then it is possible that there is to be a relationship between the entities corresponding to A and B. The type of the relationship is 1-1 if A.a is a key and otherwise 1-N. The attributes of B.b will become identifying attributes of the entity corresponding to B.

Example 3.4:

DEPARTMENT(Dept#, Floor)
EMPLOYEE(Emp#, Dept#, Salary)
EMPLOYEE.Dept # << DEPARTMENT.Dept#

In this example, a relationship (which may be called Works_in) between Employee and Department is introduced.

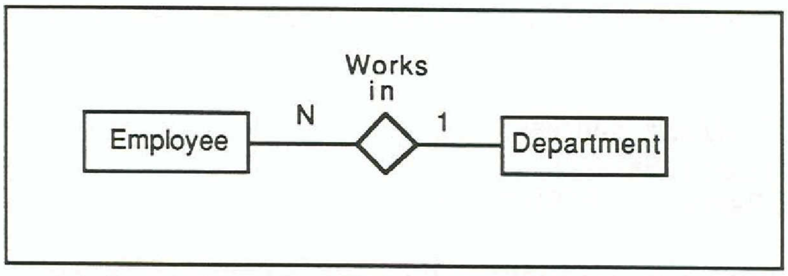


Figure 5.3 Result of EMPLOYEE.Dept # << DEPARTMENT.Dept#.

Example 3.5:

CAPITAL(Name)
COUNTRY(Country_name, Capital)
COUNTRY.CAPITAL << CAPITAL.NAME

In this example a relationship (which may be called Has) is introduced between the entities Capital and Country.

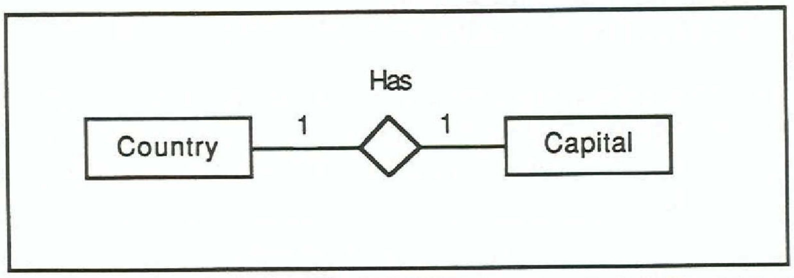


Figure 5.4 Result of COUNTRY.CAPITAL << CAPITAL.NAME.

Case 3: Introduction of a new entity and a subtype relation. This case can be divided into the following two sub cases:

Case 3a: Let I = A.a << B.b be an inclusion dependency. If A corresponds to an entity, A.a is a key, B corresponds to an entity and B.b is a candidate key or a non key then it is possible that the following are to be introduced: a new entity corresponding to B.b, a subtype relation from the entity corresponding to A to this new entity and a relationship between the new entity and the entity corresponding to B. The type of the relationship is 1-1 if B.b is a key and 1-N otherwise. The attributes B.b will become identifying attributes of the new entity.

Example 3.6:

COUNTRY(Name, Capital)
LARGE_CAPITAL(Name, Population)
LARGE_CAPITAL.Name << COUNTRY.Capital

In this example a new entity Capital is introduced corresponding to COUNTRY.Capital, a subtype relation from Large_capital to Capital and a relationship between Country and Capital.

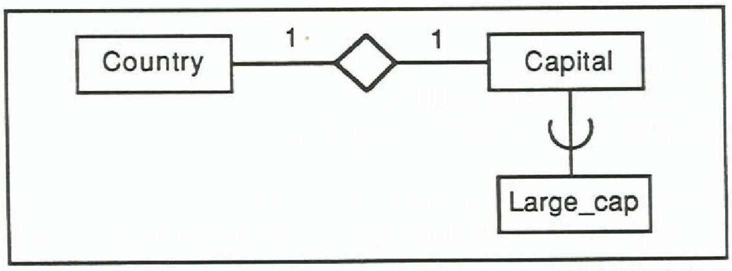


Figure 5.5 Result of LARGE_CAPITAL.Name << COUNTRY.Capital.

Case 3b: Let I = A.a << B.b be an inclusion dependency. If A corresponds to an entity, A.a is a key, B corresponds to a relationship and B.b is a non key contained in the primary key of B then it is possible that the following are to be introduced: a new entity corresponding to B.b and a subtype relation from the entity corresponding to A to this new entity. The new entity will be an entity participating in the relationship corresponding to B. The attributes B.b will become identifying attributes of the new entity.

Example 3.7:

```
DEPARTMENT_PROJECT(Dept#, Proj#, Budget)
HIGH_RISK_PROJECT(Proj#, Max_cost)
HIGH_RISK_PROJECT.Proj# << DEPARTMENT_PROJECT.Proj#
```

In this example the following are introduced: a new entity corresponding to DEPARTMENT_PROJECT.Proj# and a subtype relation from High_risk_project to Project. Project will be an entity participating in Department_project.

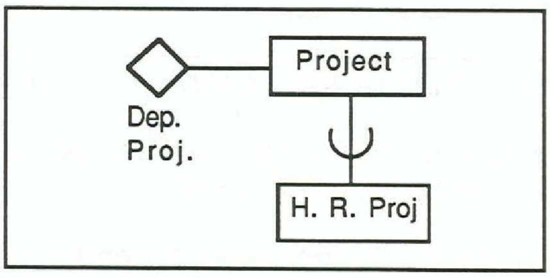


Figure 5.6 Result of HIGH_RISK_PROJECT.Proj# << DEPARTMENT_PROJECT.Proj#.

Case 4: Introduction of a new entity and its connections to existing entities and relationships. This case can be divided into the four subcases:

Case 4a: Let I = A.a << B.b be an inclusion dependency. If A corresponds to a relationship, A.a is a non key contained in the primary key of A, B corresponds to an entity and B.b is a candidate key or a non key then it is possible that there is to be a new entity corresponding to B.b. The new entity will be an entity participating in the relationship corresponding to A and there will be a relationship between the new entity and the entity corresponding to B. The type of the latter relationship is 1-1 if B.b is a key and 1-N otherwise. The attributes B.b will give rise to identifying attributes of the new entity.

Example 3.8:

```
CAPITAL(Name, Country)
MEMBERSHIP(Country, Organization)
MEMBERSHIP.Country << CAPITAL.Country
```

In this example a new entity Country is introduced, which will become an entity participating in Membership. There will also be a relationship between Country and Capital.

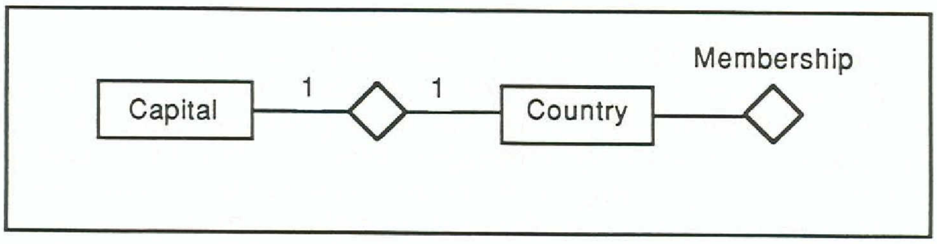


Figure 5.7 Result of MEMBERSHIP.Country << CAPITAL.Country.

Case 4b: Let I = A.a << B.b be an inclusion dependency. If A corresponds to an entity, A.a is a candidate key or a non key, B corresponds to an entity and B.b is a candidate key or a non key then it is possible that there is to be a new entity corresponding to B.b, a relationship between the entity corresponding to A and the new entity and a relationship between the entity corresponding to B and the new entity. The type of the first relationship is 1-1 if A.a is a key and otherwise 1-N. The type of the second relationship is 1-1 if B.b is a key, otherwise 1-N. The attributes B.b will become identifying attributes of the new entity.
Example 3.9:

COUNTRY(Name, Currency)
INFLRATE(Currency, Year, Rate)
INFLRATE.Currency << COUNTRY.Currency

In this example the following are introduced: a new entity Currency corresponding to COUNTRY.Currency, a relationship between Country and Currency and a relationship between Inflrate and Currency. Currency will get the identifying attribute name.

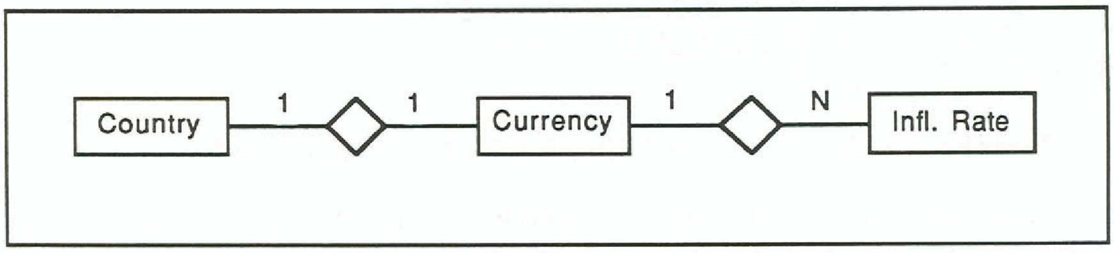


Figure 5.8 Result of INFLRATE.Currency << COUNTRY.Currency.

Case 4c: Let I = A.a << B.b be an inclusion dependency. If A corresponds to an entity, A.a is a candidate key or a non key, B corresponds to a relationship and B.b is a non key contained in the primary key of B then it is possible that there is to be a new entity corresponding to B.b and a relationship between the entity corresponding to A and the new entity. The new entity will be an entity participating in the relationship corresponding to B. The type of the new relationship is 1-1 if A.a is a key and otherwise 1-N. The attributes B.b will become identifying attributes of the new entity.

Example 3.10:

DEPARTMENT_PROJECT(Dept#, Proj#, Budget)
EQUIPMENT(E#, Proj#)
DEPARTMENT(Dept#)
EQUIPMENT.Proj# << DEPARTMENT_PROJECT.Proj#

In this example a new entity Project, corresponding to DEPARTMENT_PROJECT.Proj# and a new relationship (which may be called Belongs_to) between Equipment and Project are introduced. Project will be an entity participating in Department_project (see Figure 5.9).

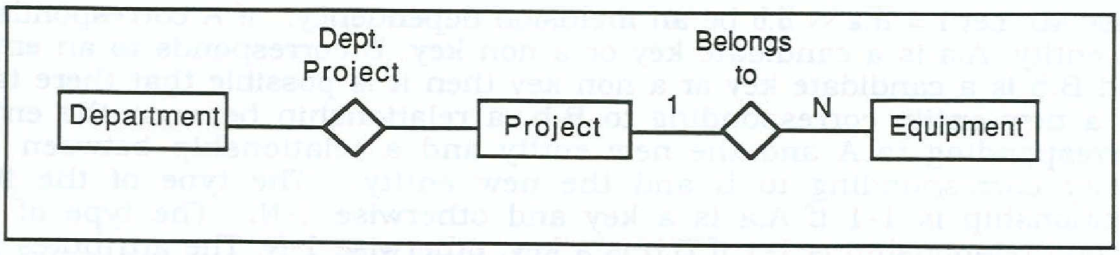


Figure 5.9 Result of EQUIPMENT.Proj# << DEPARTMENT_PROJECT.Proj#.

Case 4d: Let I = A.a << B.b be an inclusion dependency. If A corresponds to a relationship, A.a is a non key contained in the primary key of A, B corresponds to a relationship and B.b is a non key contained in the primary key of B then it is possible that there is to be a new entity corresponding to B.b. The new entity will be an entity participating in both the relationship corresponding to A and the relationship corresponding to B. The attributes B.b will become identifying attributes of the new entity.

Example 3.11:

```
DEPARTMENT_PROJECT(Dept#, Proj#, Budget)
EMPLOYEE_PROJECT(Emp#, Proj#, Hours)
DEPARTMENT(Dept#)
EMPLOYEE(Emp#)
EMPLOYEE_PROJECT.Proj# << DEPARTMENT_PROJECT.Proj#
```

In this example a new entity Project, corresponding to DEPARTMENT_PROJECT.Proj# *is* introduced. Project will be an entity participating in both Department_project and Employee_project.

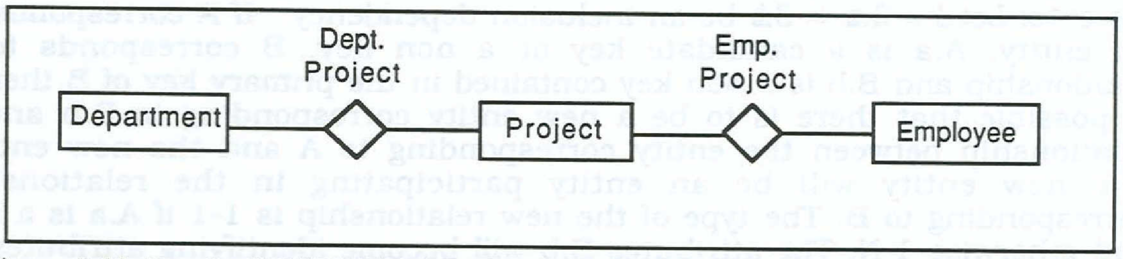


Figure 5.10 Result of EMPLOYEE_PROJECT.Proj# << DEPARTMENT_PROJECT.Proj#.

STEP4 - Attributes which do not occur in some inclusion dependency are handled here. This will be done in two substeps:

a) If R is a ternary relation corresponding to a relationship and $k_1,\ldots,k_n$ are attributes contained in the primary key of R, which have not been handled in step 3 then these attributes will give rise to one or more new entities, which will become entities participating in the relationship corresponding to R.

Example 4.1:

```
COUNTRY(Name, Capital)
MEMBERSHIP(Country, Organization)
MEMBERSHIP.Country << COUNTRY.Name
```

After step 3 the schema in figure 5.11 is obtained and after step 4a the schema of figure 5.12.

b) If a is an attribute of a relation R and a has not been handled in step 3 or 4a then a will give rise to an attribute of the entity or relationship corresponding to R. Continuing the example in step 4a above the attribute **COUNTRY.Capital** will give rise to an attribute capital of the entity Country, see fig. 5.13.

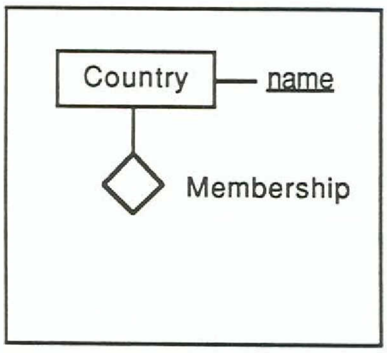


Figure 5.11 After step 3.

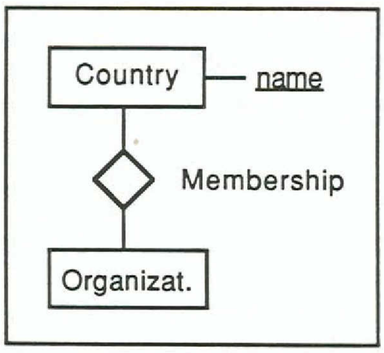


Figure 5.12 After step 4a.

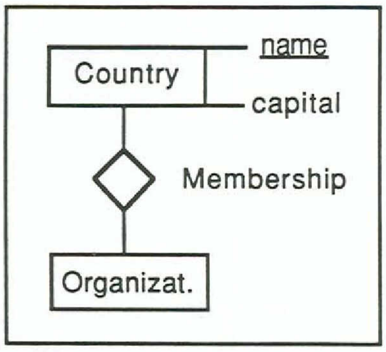


Figure 5.13 After step 4b.

5.3 SUBSUMPTION OF INCLUSION DEPENDENCIES

Under certain circumstances an inclusion dependency will not give rise to any additions to a conceptual schema. The reason for this is that an inclusion dependency may be "subsumed" (i.e. made redundant) by other inclusion dependencies.

An inclusion dependency I = **A.a** << **C.c** is said to be subsumed by two other inclusion dependencies J and K if either (a) or (b) below holds.

(a) J = **B.b** << **C.c** and
K = **A.a** << **B.b** hold and
B.b and C.c are keys

(b) J = **C.c** << **D.d** and
K = **A.a** << **D.d** hold and
C.c is a non key and D.d is a key

An illustration of (a) above is the following:

PERSON(<u>SS#</u>, Address)
EMPLOYEE(<u>SS#</u>, Salary)
SECRETARY(<u>SS#</u>, Computer#)
EMPLOYEE.SS# << PERSON.SS#
SECRETARY.SS# << EMPLOYEE.SS#
SECRETARY.SS# << PERSON.SS#

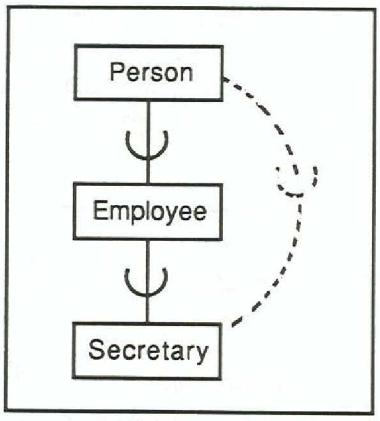


Figure 5.14 Superfluous link.

In this example the last inclusion dependency is subsumed by the previous two. Had the last inclusion dependency been treated, case 1 would have applied giving rise to a subtype from Secretary to Person. However, this subtype relation is superfluous since there will be subtype relations from Secretary to Employee and from Employee to Person. The last inclusion dependency in the example above can, therefore, be disregarded when building the conceptual schema.

Example: An example of (b) above is the following:

COUNTRY(<u>Name</u>, Capital)
MEMBERSHIP(<u>Country, Organization</u>)
POPULATION(<u>Country, Year</u>, Population)
MEMBERSHIP.Country << COUNTRY.Name
POPULATION.Country << COUNTRY.Name
MEMBERSHIP.Country << POPULATION.Country

In this example the last inclusion dependency is subsumed by the two previous. Had this inclusion dependency been handled, case 4a would have been applicable and a new entity corresponding to POPULATION.Country would have been introduced. Such an entity is, however, superfluous since an entity Country corresponding to the relation COUNTRY already exists. The connection between this entity and the relationship Membership is given by the first inclusion dependency above. Consequently, we can disregard the last inclusion dependency. Figure 5.15 illustrates this situation.

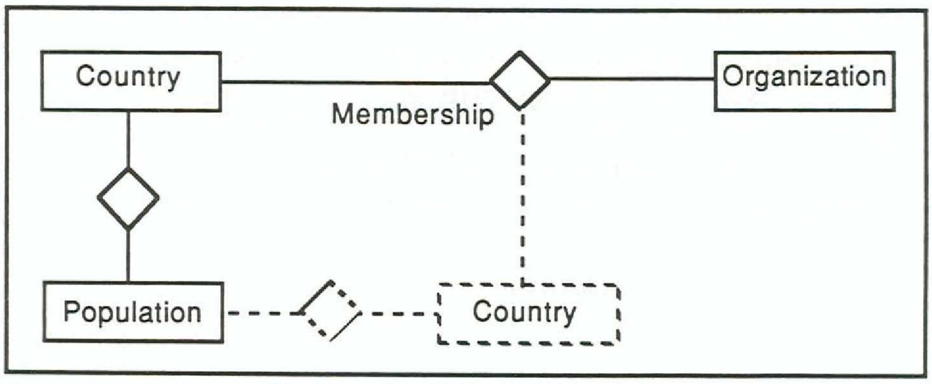


Figure 5.15 Superfluous entity.

5.4 HANDLING AMBIGUOUS CASES

Every inclusion dependency which is not subsumed by other inclusion dependencies is to be handled by one of the four cases above. This will result in additions to the preliminary conceptual schema produced by step 2 of the method. Sometimes a situation in which more than one case may be a candidate for handling an inclusion dependency may arise. To solve this problem the user is asked to choose the appropriate case. The table depicted in fig. 5.16 shows all of the various types of inclusion dependencies that may occur and the cases which can be used for handling these. As an example, illustrating how to interpret the table, the second row of the table is to be read as "if I is an inclusion dependency whose left hand side is a primary key and whose right hand side is a candidate key then I can be handled by either case 1 or case 3". Examples 3.2 and 3.6 above show that there are two alternative ways of handling an inclusion dependency of that type.

Type of Inclusion Dependency	Possible Cases
Primary Key << Primary Key	1
Primary Key << Candidate Key	1 , 3
Primary Key << Non Key	3
Candidate Key << Primary Key	1 , 2
Candidate Key << Candidate Key	1 , 2 , 3 , 4
Candidate Key << Non Key	3 , 4
Non Key << Primary Key	2
Non Key << Candidate Key	2 , 4
Non Key << Non Key	4

Figure 5.16 Table showing interaction between keys, inclusion dependencies and modelling constructs.

Inspection of the above table leads to the observation that only those inclusion dependencies involving candidate keys have more than one possible way of being handled. The reason for the complication caused by candidate keys is that a candidate key may indicate the presence of a corresponding entity. An example illustrating this is the relation COUNTRY(Name, Capital), where the candidate key Capital may give rise to an entity of its own, i.e. something about which we want to collect information. In other cases a candidate key does not give rise to an entity. An example is the relation EMPLOYEE(Emp#,SS#,Salary), where the candidate key SS# is only an alternative identifier of the entity Employee.

5.5 AN EXAMPLE

Relational database schema given as input:

COUNTRY(Name,Capital,Currency)	(primary)
CURRENCYVALUE(Currency, Value_in_$)	(primary)
CITYPOPULATION(City, Year, Population)	(primary)
MEMBERSHIP(Country, Organization, Entry_date)	(ternary)
EXPORT(Supplier, Consumer, Amount)	(secondary)
COMPANY(Name, Country, Revenues)	(primary)
EUROPEAN(Country, Population)	(primary)

CURRENCYVALUE.Currency << COUNTRY.Currency	(3a)
COUNTRY.Capital << CITYPOPULATION.City	(4b)
MEMBERSHIP.Country << COUNTRY.Name	(2a)
EXPORT.Supplier << COUNTRY.Name	(2a)
EXPORT.Consumer << COUNTRY.Name	(2a)
COMPANY.Country << COUNTRY.Name	(2b)
EUROPEAN.Country << COUNTRY.Name	(1)

To the right of each relation its classification is given. The number to the right of each inclusion dependency indicates by which case of step 3 it is to be handled. The following figures show the intermediate schemas and the final schema.

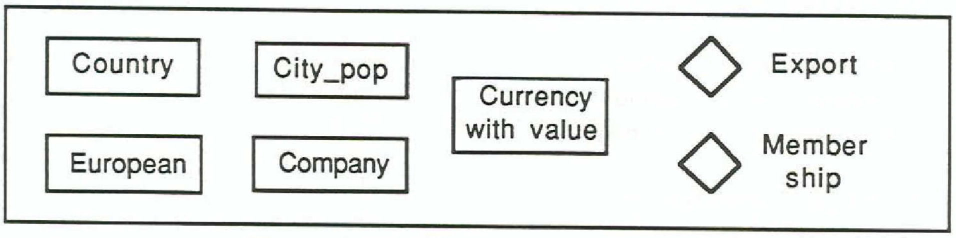


Figure 5.17 The schema after step 2.

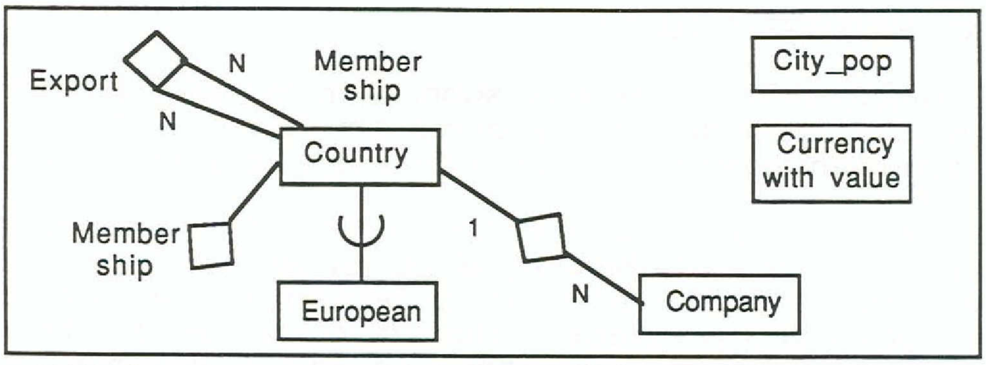


Figure 5.18 The schema after having handled inclusion dependencies of type 1 and 2.

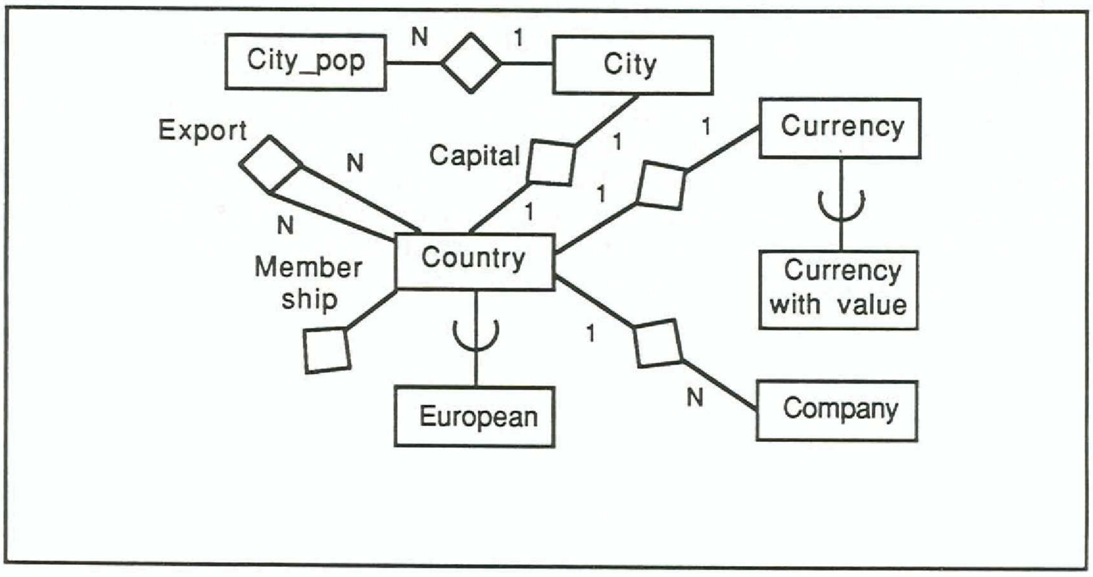


Figure 5.19 The schema after step 3.

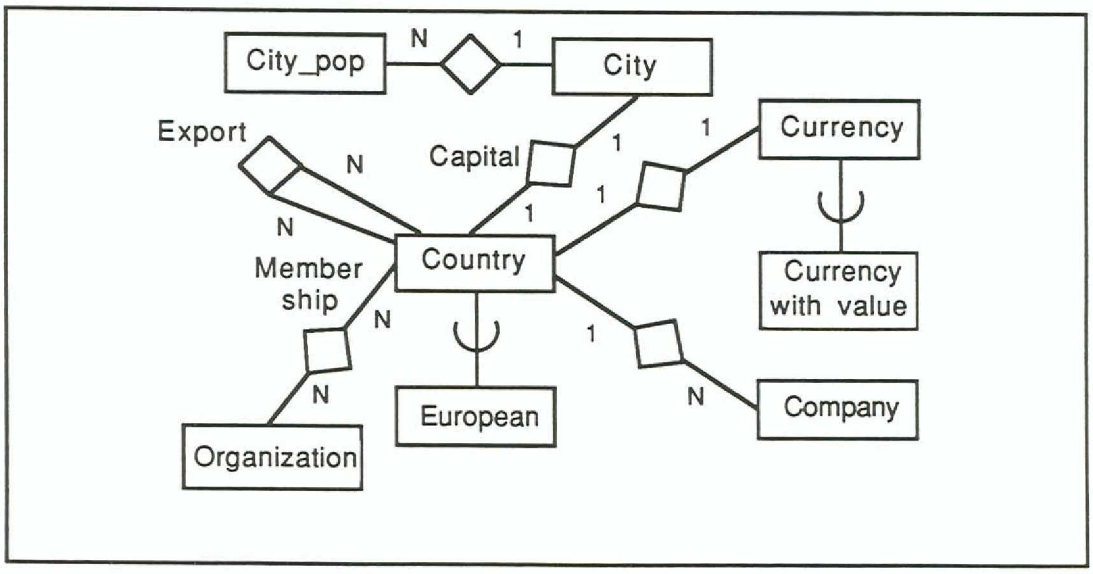


Figure 5.20 The final schema.

5.6 CONCLUDING EVALUATION

The accessibility of the information supplied by inclusion dependencies to the algorithm simplifies the mapping problem considerably. This simplification can, however, be viewed both as a strength and a weakness of the method.

On the one hand, it causes the problem of creating entities and relations between them to be a trivial one since a great majority of the information used to create relationships between entities is contained in the given inclusion dependencies. The relationship between input to and output from the algorithm almost has a one to one correspondence. This renders the translation process easy, clean and accurate.

The disadvantage of providing inclusion dependencies is that a large part of the solution is supplied to the algorithm. Arriving at a conceptual model from something which is almost a conceptual model is not all that remarkable.

A way to alleviate this problem is to generate these dependencies. Suggestions for all potential inclusion dependencies of a database can be generated mechanically by searching through the extension (the tuples) of the database. Mechanizing the generation of inclusion dependencies allows for less user interaction while maintaining the high degree of lucidity and accuracy of the mapping algorithm.

6

A COMBINED APPROACH

6.1 INTRODUCTION

A combined method using inclusion dependencies as well as similarity in attribute names to obtain the information necessary to map relations in a relational database to a conceptual model is presented here. In contrast to the two previously described methods, neither inclusion dependencies nor identical attribute names are required to be included in the relational database in order for the algorithm to establish connections between two or more relations in the database. Instead, potential inclusion dependencies, based on identical or similar attribute names, are suggested for the user. The existence of a proposed inclusion dependency is then either confirmed or denied by the user. Inclusion dependencies deemed to hold true for the database in question, are added on to the given database and used to establish connections between the relations of the database.

A dictionary of synonyms is used to infer similarity in attribute names. This dictionary consists of a number of entries. Each entry is made up of a list of words with similar meanings. Every time a proposed inclusion dependency is confirmed or a new inclusion dependency is entered, the dictionary is updated with the new synonyms. This renders the method dynamic as part of the information it uses, to create the conceptual schema, is always growing in size.

The data initially available to the mapping process consists of the relations of a database. The relations of the schema are always in the third normal form and can be given with or without instances.

6.2 A TRANSFORMATION STRATEGY IN SEVEN STEPS

The proposed algorithm using both inclusion dependencies and attribute name similarity to map a relational database to an ER schema has been implemented in MAKEMODEL3, a program written in Prolog. Input to the program is a relational schema given with or without detail information (instances) and a dictionary of synonyms. For each relation, a relation name, key and none key attributes as well as the domain[5] of all attributes are specified.

The relations are represented as Prolog facts. These facts are, then, processed by a seven step procedure which creates an extended Entity-Relationship semantic data model containing subclasses. In **step1**, attributes of different relations, having the same domain, same name or whose names are synonyms contained in the dictionary, or where the instances of one attribute make up a subset of the other, are used to generate potential inclusion dependencies. In **step2** candidate keys are handled and, if needed, relations that are to become new entities are created as well as new inclusion dependencies used to connect these new

[5] A domain is a set of nondecompasable values. It is the smallest unit of data in the relational model. For example, the domain of shipment quantities is the set of all integers greater than zero and less than 10,000.

relations to existing ones. In **step3**, the relations of the database are classified. Entities and relationships are created in **step4**. In **step5**, connections between entities that have a sub/superclass relationship to each other are established. In **step6**, all remaining connections are handled based on the inclusion dependencies. In **step7**, the mappings of the functional dependencies between the entities of binary relationships are set up.

A detailed account of the seven step procedure used by MAKEMODEL3 is given below:

STEP1 - potential inclusion dependencies are searched out and suggested for the user. This is done in two phases. During the first phase all potential inclusion dependencies, for the given relations, are sought out based on the type information provided for the database. The second phase is divided into two cases. The first case is that of a database for which both type and detail information is given. In this case the inclusion dependencies are further deduced from the extensions of the database. The second case is that where the database is given on a type level only. Here, a dictionary of synonyms is used to further limit the set of likely candidates for inclusion dependencies. In both cases, the user is asked to confirm or deny the suggested inclusion dependencies. In the second case the user is asked to provide any remaining inclusion dependencies not suggested by the system. After the two phases have been carried out, the dictionary of synonyms is updated with any new information.

Phase I - The domains of all attributes of all relations are examined. For each domain, all attributes, having that domain, of some relation are paired with an attribute, of the same domain, of another relation. The set of Relation/Attribute pairs (R1,A1,R2,A2) makes up the set of all possible inclusion dependencies for the database.

Phase II - The set of potential inclusion dependencies produced in phase I is further reduced here.

Case1 - A complete set of inclusion dependencies is mechanically arrived at here. Two actions are carried out for this case.
action1 - The extensions of the database are examined with the set of potential inclusion dependencies. For each Relation/Attribute pair, the extensions of the two attributes are examined in order to ascertain if one is a subset of the other. The resulting set is a more limited set of the original set produced in phase I but is a set which also includes all potential inclusion dependencies.
action2 - This set is then shown to the user who is, then, asked if the inclusion dependencies are valid. Each valid inclusion dependency is added to the database. The attribute pairs contained in the set of inclusion dependencies, accepted as valid, are gathered to update the synonym dictionary.

case2 - An incomplete set of inclusion dependencies is arrived at mechanically which the user must, then, complete. Three actions are carried out for this case.

action1 - The names of the attributes in the potential Relation/Attribute pairs are checked. If the names of the two attributes contained in the pair are identical, the pair is placed in a set of likely candidates. When the names are different, a synonym dictionary is checked. If one of the names is found to be contained in the dictionary, the pair is included in the 'likely' set.
action2 - This set of likely inclusion dependencies is shown to the user who is then asked if the inclusion dependencies are valid. Each valid inclusion dependency is added to the database.
action3 - The user is now asked to provide all the remaining inclusion dependencies that were missed by the previous action. These are also added to the database for processing. The attribute pairs contained in the entered inclusion dependencies are gathered and the synonym dictionary is updated.

STEP2 - Candidate and none keys are handled and, when appropriate, new relations and inclusion dependencies are created which, in the future, will give rise to new entities. Four actions are carried out in this step.

action1 - The candidate keys of each relation in the database are examined. For each candidate key which is also found to be on the right hand side of an inclusion dependency, a question is put to the user asking if the attribute, making up this candidate key, should be made into an entity. If the user answers 'yes', a relation is created out of the candidate key. This new relation will get a primary key with the same name as the relation. The candidate key is deleted from the relation and is added as a none key to the relation. The original inclusion dependency is deleted and two new inclusion dependencies are added in order to connect the new entity with the two entities which will result from the relations corresponding to the right and left hand sides of the original inclusion dependency.

Example 2.1:

CINEMA($\underline{\text{Name}}$,$\underline{\text{Film}}$)
THRILLER($\underline{\text{Name}}$)
THRILLER.Name << CINEMA.Film

After action1 the changes would result in the following database:

CINEMA ($\underline{\text{Name}}$,Film)
THRILLER($\underline{\text{Name}}$)
FILM($\underline{\text{Film}}$)
THRILLER.Name << FILM.Film
CINEMA.Film << FILM.Film

action2 - The candidate keys of each relation in the database are examined. If a candidate key is found to be on the left side of an inclusion dependency, and the attribute on the right hand side of the inclusion dependency is a key, a question is put to the user asking if there prevails a subclass superclass relationship between the entities corresponding to the two relations. If the user answers 'yes', the candidate key is changed to become the primary key of the relation. The two relations are then ready to be processed as having a sub/superclass relationship to each other. If the answer is 'no', the user is asked whether there prevails a subclass supercalss relationship between an entity corresponding to the candidate key and the entity corresponding to the relation on the right hand side of the inclusion dependency. If that is the case a new relation is created out of the candidate key and inclusion dependencies are set up to connect the new entity with the entities which will result from the relations corresponding to the right and left hand sides of the inclusion dependency. If none of these subclass superclass relationships exist, the inclusion dependency will, later, give arise to an ordinary relationship.

Example 2.2:

```
PERSON(SS_num)
CUSTOMER(Cust_num,SS_num)
CUSTOMER.SS_num << PERSON.SS_num
```

Question asked in action2:
Is CUSTOMER a subclass of PERSON?

After action2 the changes would result in the following database:

```
PERSON(SS_num)
CUSTOMER(SS_num,CUST_num)
CUSTOMER.SS_num << PERSON.SS_num
```

action3 - The primary key of each relation which also is a relation contained on the right hand side of an inclusion dependency is examined. If an attribute contained on the right hand side of an inclusion dependency is a subset of the primary key of the relation contained on the right hand side of the inclusion dependency, a new entity corresponding to the attribute on the right hand side of the inclusion dependency, is created. The original inclusion dependency is deleted and two new inclusion dependencies are added in order to connect the new entity with the two entities which will result from the relations corresponding to the right and left hand sides of the original inclusion dependency.

Example 2.3:

```
RENT(Store,Film)
CUSTOMER(Cust_num, Store)
CUSTOMER.Store << Rent.Store
```

After action3 the changes would result in the following database:

RENT(Store,Film)
CUSTOMER(Cust_num, Store)
STORE(Store)
CUSTOMER.Store << STORE.Store
RENT.Store << STORE.Store

action4 - The none key attributes of each relation in the database are examined. For each such attribute which is also found to be on the right hand side of an inclusion dependency, a new relation is created. This new relation will get a primary key with the same name as the relation. Two inclusion dependencies are added in order to connect the new entity with the two entities which will result from the relations corresponding to the right and left hand sides of the inclusion dependency.

STEP3 - The relations of the database schema are classified into four types: primary (PR), secondary (SR) and two types of ternary (TR1 and TR2) relations.

PR -	Relation with the property that no subset of its primary key occurs on the left hand side of an inclusion dependency. (The entire key may, however, occur on the left hand side of an inclusion dependency.)
SR -	Relation whose primary key is equal to the concatenation of the left hand sides of at least two inclusion dependencies.
TR -	Relation such that some attribute of its primary key occurs on the left hand side of an inclusion dependency and some other attribute of the primary key does not occur on the left hand side of an inclusion dependency.

The primary relations will correspond to entities in the conceptual schema, the secondary to relations and the ternary relations to either entities or relationships. TR1s will correspond to weak entities. A weak entity is an entity whose key identifier is ID dependent on another entity's key identifier. TR2s will give rise to relationships. It is via a user question that it is determined if the TR relation will be an entity or a relationship.

Example 3.1:

PERSON(Soc_Sec, Name,Address)
SPECTATOR(SS_num, Seat_Num)
WATCH(SS, Movie_Title)
MOVIE(Title)
SPECTATOR.SS_num << Person.Soc_Sec
WATCH.SS << SPECTATOR.SS_num
WATCH.Movie_Title << MOVIE. Title

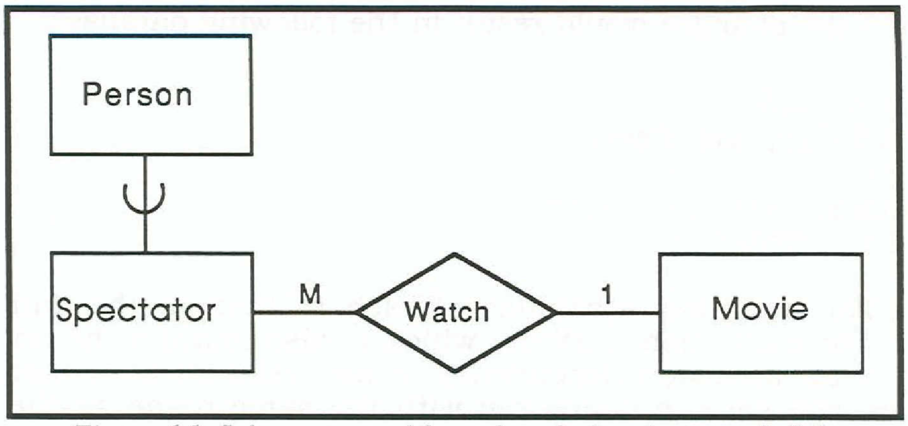


Figure 6.1 Schema created from the relations in example 3.1.

In this example **Person**, **Spectator** and **Movie** are primary relations, while **Watch** is a secondary relation.

Example 3.2:

> VIDEOSTORE(<u>Store#</u>, Address)
> DEPT(<u>Store#,Dept#</u>,Record_Name)
> FILM(<u>Film_Title</u>)
> LEND(<u>Store#,Film_Title,Customer</u>)
> DEPT.Store# << VIDEOSTORE.Store#
> LEND.Store# << VIDEOSTORE.Store#
> LEND.Film_Title << FILM.Film_Title

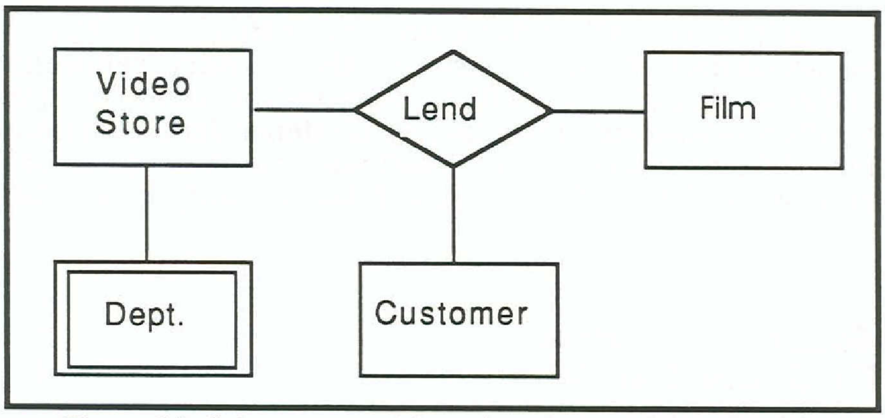


Figure 6.2 Schema created from the relations in example 3.2.

In the above example DEPT and LEND are ternary relations. As shown in figure 6.2, LEND will become a relationship while DEPT will become a weak entity.

STEP4 - Entities and relationships corresponding to the relations of the database are created.

Entities are created from PR relations. TR1 relations map to weak entities. SR and TR2 relations give rise to relationships. Attributes contained in the primary key of TR2 relations to which corresponding entities do not exist are mapped to new entities (the entity customer in example 3.2 above is an illustration of such a case).

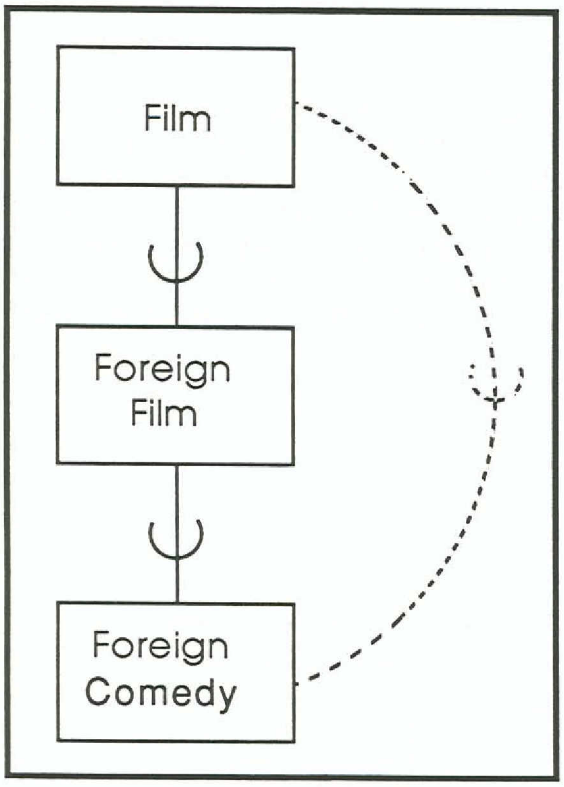
STEP5 - All subclass/superclass relationships are handled here.

If the attribute(s) appearing on the left hand side of an inclusion dependency constitute the primary key of the entity corresponding to the relation named on the left hand side of the inclusion dependency and the attribute(s) appearing of the left hand side of the same inclusion dependency constitute a key of the relation appearing on the right hand side, a sub/super class relationship is established between the two entities.

Example 5.1:

```
FILM(Film_Name,Director,Year)
FOREIGN_LANG_FILM(Film_Name,Language)
FOREIGN_LANG_FILM.Film_Name << FILM.Film_Name
```

After all subclass/superclass relationships have been identified and generated, any indirect links between entities in a chain involving two or more levels of sub/superclass relationships are deleted. Figure 6.3 below illustrates such a subsumption.



The link between Foreign Comedy and Film should not be represented explicitly in the conceptual schema.

Figure 6.3 Superfluous indirect link.

STEP6 - All inclusion dependencies are handled here to establish connections between the data structures of the ER schema. When a relationship is created by other means than by mapping a relation in the database to a relationship, the relationship will not have a name. In such cases, the user will be asked to furnish a name for the relationship.

The following connections are identified and set up here:

- Relationships arising from SR and TR2 relations. These relationships were generated from relations appearing on the left hand sides of inclusion dependencies. The entities generated from the relations on the right hand sides of the same inclusion dependencies will become the entities connected to the relationship.

Example 6.1:

```
CUST(Cust_num,Name,Addr)
FILM(Name)
RENT(Customer,Film_Title)
RENT.Customer << CUST.Cust_num
RENT.Film_Title << FILM.Name
```

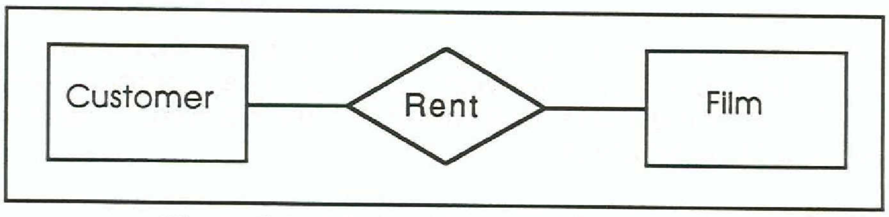


Figure 6.4 Relationship created from an SR.

- Relationships created from FKA connections between two entities. An FKA (Foreign Key Attribute) is a none key attribute of a relation which has been mapped to an entity, such that there is an inclusion dependency from the key of another relation to the attribute. In this case the relation on the left hand side of the inclusion dependency must also be mapped to an entity. The relationship between the two entities will be an entirely new relationship. The user will be required to give a name to this relationship.

Example 6.2:

```
MEMBER(Cust_num,Videostore)
VIDEOSTORE(Name)
MEMBER. Videostore << VIDEOSTORE.Name
```

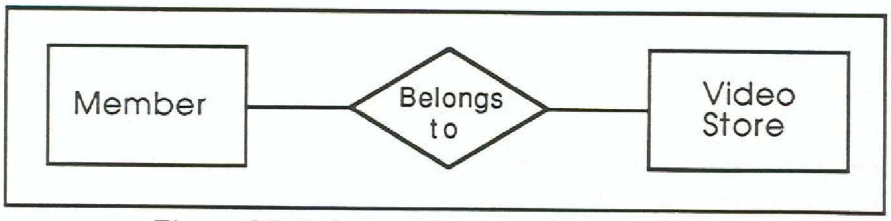


Figure 6.5 Relationship created from an FKA.

- Connections between an entity and a relationship through an FKA. An inclusion dependency from a none key of a relation corresponding to a relationship to a key of a relation corresponding to an entity will result in a connection between the entity and the relationship.

Example 6.3:

```
RENT(Cust.Film,Store_num)
CUSTOMER(Name)
FILM(Name)
VIDEOBOUTIQUE(Store_num)
RENT.Store_num << VIDEOBOUTIQUE.Store_num
```

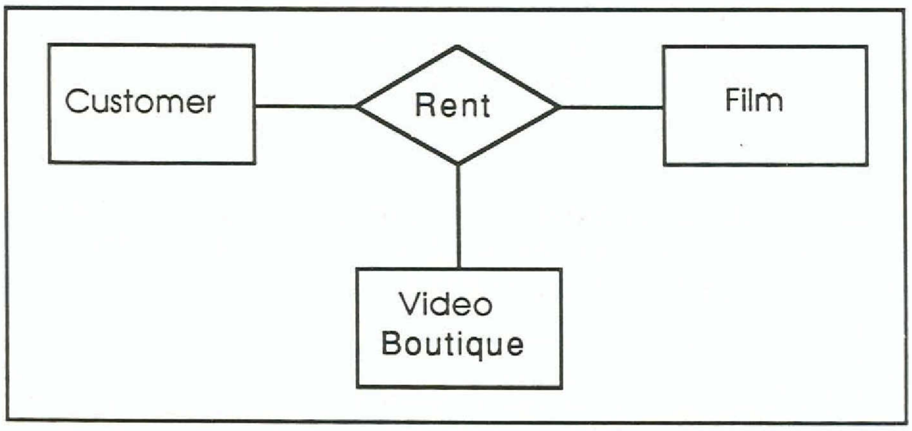


Figure 6.6 An FKA connection between an entity and a relationship.

- Connections between weak entities and the entities on which these are ID dependent. A weak entity corresponding to a relation found on the right hand side of an inclusion dependency whose key contains the key of the relation, corresponding to an entity, on the left hand side of the inclusion dependency will give rise to a connection between the two entities.

Example 6.4:

```
VIDEO_COMPANY(Name)
VIDEO_STORE(Company_Name,Store_num)
VIDEO_STORE.Company_Name << VIDEO_COMPANY.Name
```

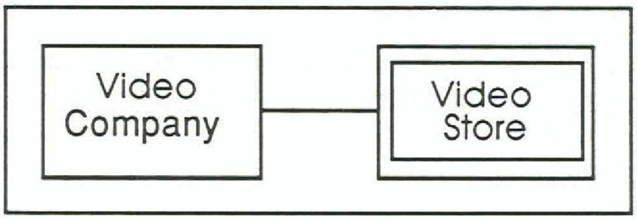


Figure 6.7 The weak entity Video store.

STEP7 - Binary relationships are identified and labeled as such. For each binary relationship, the mapping of the functional dependency between the two entities for the relationship is established. In the case of relations produced from FKA connections between two entities, the mappings are produced automatically. For binary relationships produced from SR and TR2 relations, the user is asked to specify the correct mapping.

Mappings produced automatically are done in the following way: If the FKA in the original relation, that is the attribute on the left hand side of the inclusion dependency, is a none key, the relation gets a mapping of many to one. If the FKA of the original relation is a candidate key, the mapping becomes one to one. The motivation for this is that the instances of none key attributes are not necessarily unique for the relation, whereas candidate key attributes are unique. Several instances of none key attributes can, therefore, correspond to one key attribute, but only one key attribute can correspond to another key attribute of two different relations on two sides of an inclusion dependency.

Example 7.1:

The relational database consisting of the two relations Sales_person and Store, complete with instances and supplemented with the inclusion dependency: SALES_PERSON.Store << STORE.Name,

Sales_person

Emp_num	Store
123456	Video_butique
234567	Video_butique
345678	Video_land

Store

Name
Video_butique
Video_house
Video_corner
Video_land
Video_palace

will give rise to the following conceptual schema:

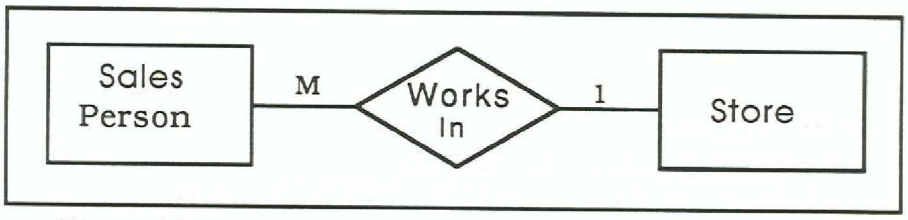


Figure 6.8 The mapping created automatically from example 7.1.

For binary relationships produced from SR relations the user must chose between one of the following four types of mappings:
- one to many (1,M)
- many to one (M,1)
- one to one (1,1)
- many to many (M,M)

Example 7.2:

```
VIDEO_STORE(Name)
CASSETTE(Serial_number)
OWNS(Sore,Cassette)
OWNS.Store << VIDEO_STORE.Name
OWNS.Cassette << CASSETTE.Serial_number
```

The user is asked to choose one of the above mappings to describe the functional dependency between the entity Video_store and the entity Cassette in the relationship Owns. The user chooses, for example, (1,M). This results in the following mapping:

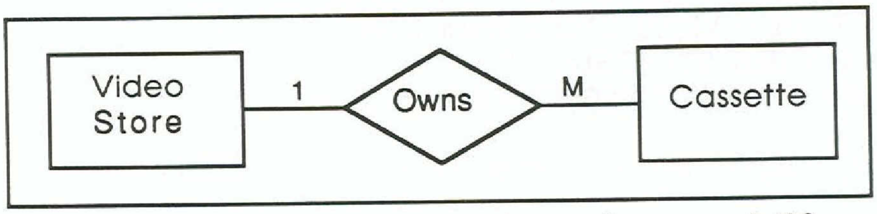


Figure 6.9 The mapping given by the user from example 7.2.

6.3 AN EXAMPLE

Relational database schema given as input:

```
VIDEO_STORE(Store_Num,Addr,Company_Name)
VIDEO_COMPANY(Name,Yearly_Profits)
DEPARTMENT(Dept_Name,Store_Num,Product)
FILM(Title,Director,Year)
BUY(Video_Company,Wholesale_Company)
EMPLOYEE(Emp_Num,Name)
WORKS_IN(Emp_Num,Store_Num)
CUSTOMER (Cust_Num,Address,Name)
RENT(Cust_Num,Film_Title,Store_Num)
CD_RECORD(Name)
SELLS(Cust,Record,Store)
MEMBER(Cust_Num,Bonus_Points)
MUSICAL(Film_Ser_Num,Film_Title)
COMEDY(Film_Ser_Num,Film_Title)
THRILLER(Film_Ser_Num,Film_Title)
REQUEST(From_Store,To_Store)
```

During **step1**, inclusion dependencies for the database, shown above, are suggested and/or requested from the user. It is assumed that attributes can have one of two domains: integer, string. It is also assumed that the synonym dictionary contains the following entries:

- company, company_name, name
- from_store, store_num, store_number, to_store
- cust,cust_num

- film, film_name, film_title, movie_title, title
- store, store_num

MAKEMODEL3 will then give the following correct suggestions (among other incorrect ones) for the domain integer based on either identical or synonymous names of attributes.

```
MEMBER.Cust_Num >> CUSTOMER.Cust_Num
RENT.Cust_Num >> CUSTOMER.Cust_Num
SELLS.Cust << CUSTOMER.Cust_Num
DEPARTMENT.Store_Num << VIDEO_STORE.Store_Num
WORKS_IN.Emp_Num << EMPLOYEE.Emp_Num
RENT.Store_Num << VIDEO_STORE.Store_Num
REQUEST.From_Store << VIDEO_STORE.Store_Num
REQUEST.To_Store << VIDEO_STORE.Store_Num
SELLS.Store << VIDEO_STORE.Store_Num
WORKS_IN.Store_Num << VIDEO_STORE.Store_Num
```

It will then suggest the following inclusion dependencies for the string domain.

```
COMEDY.Film_Title << FILM.Title
MUSICAL.Film_Title << FILM.Title
RENT.Film_Title << FILM.Title
THRILLER.Film_Title << FILM.Title
VIDEO_STORE.Company_Name << VIDEO_COMPANY.Name
```

The following inclusion dependencies are then provided by the user.

```
BUY_FROM.Video_company >> VIDEO_COMPANY.Name
SELLS.Record >>CD_RECORD.Name
```

In **step2** the candidate keys are handled for the three relations in the database that have candidate keys. The following questions are asked.

```
Is COMEDY a subclass of FILM ?
yes

Is MUSICAL a subclass of FILM ?
yes

Is THRILLER a subclass of FILM ?
yes
```

After this the candidate keys are made into primary keys for these three relations.

All relations are classified in **step3**. The user is requested to specify whether the TR relations should be entities or relationships.

```
DEPARTMENT could be either an entity or a relationship,
Please specify which it should be by typing an e or an r.
e
BUY could be either an entity or a relationship,
Please specify which it should be by typing an e or an r.
r
```

The relations are classified in the following way:

```
rel('VIDEO_COMPANY',pr).
rel('VIDEO_STORE',pr).
rel('DEPARTMENT',tr1).
rel('FILM',pr).
rel('BUY',tr2).
rel('EMPLOYEE',pr).
rel('WORKS_IN',sr).
rel('CUSTOMER',pr).
rel('RENT',sr).
rel('CD_RECORD',pr).
rel('SELLS',sr).
rel('MEMBER',pr).
rel('MUSICAL',pr).
rel('COMEDY',pr).
rel('THRILLER',pr).
rel('REQUEST',sr).
```

Entities, weak entities and relationships are created in **step4**.

```
[CD_RECORD] Entity Identifier: [Name]
[COMEDY] Entity Identifier: [Film_Ser_Num]
[CUSTOMER] Entity Identifier: [Cust_Num]
[EMPLOYEE] Entity Identifier: [Emp_Num]
[FILM] Entity Identifier: [Title]
[MEMBER] Entity Identifier: [Cust_Num]
[MUSICAL] Entity Identifier: [Film_Ser_Num]
[THRILLER] Entity Identifier: [Film_Ser_Num]
[VIDEO_COMPANY] Entity Identifier: [Name]
[VIDEO_STORE] Entity Identifier: [Store_Num]
[Wholesale_Company] Entity Identifier: [Wholesale_Company]
[[DEPARTMENT]] Weak Entity Identifier: [Dept_Name,Store_Num]
<BUY> Relationship Identifier: [Video_Company,Wholesale_Company]
<RENT> Relationship Identifier: [Cust_Num,Film_Title,Store_Num]
<REQUEST> Relationship Identifier: [From_Store,To_Store]
<SELLS> Relationship Identifier: [Cust,Record,Store]
<WORKS_IN> Relationship Identifier: [Emp_Num,Store_Num]
```

Subclass superclass relationships are created in **step5**.

```
subclass('COMEDY','FILM').
subclass('MEMBER','CUSTOMER').
subclass('MUSICAL','FILM').
subclass('THRILLER','FILM').
```

In **step6** all relationships are completed with the entities participating in the relationships. When a new relationship between two entities is created the user is asked to give it a name.

```
Please type in the name you wish to give the relationship
between the entity video_store and the entity video_company
runs
relship('BUY',['WHOLESALE_COMPANY','VIDEO_COMPANY']).
relship('RENT',['VIDEO_STORE','FILM','CUSTOMER']).
relship('REQUEST',['VIDEO_STORE']).
relship('SELLS',['VIDEO_STORE','CD_RECORD','CUSTOMER']).
relship('RUNS',['VIDEO_STORE','VIDEO_COMPANY']).
relship('WORKS_IN',['VIDEO_STORE','EMPLOYEE']).
id_dep('DEPARTMENT','VIDEO_STORE').
```

Mapping of the functional dependencies for binary relationships is done in **step7**. The user is requested to provide the correct mapping in the following way.

```
In order to assess the functional dependency between |VIDEO_STORE|
and |EMPLOYEE| in relationship <WORKS_IN>, please chose the correct
dependency listed below by typing the number 1, 2, 3, or 4.

1. There is a functional dependency from |VIDEO_STORE| to |EMPLOYEE|.
2. There is a functional dependency from |EMPLOYEE| to |VIDEO_STORE|.
3. Both of the above functional dependencies exist.
4. None of the above functional dependencies exist.

2
```

The final schema produced by MAKEMODEL3 is shown in Figure 6.10 below.

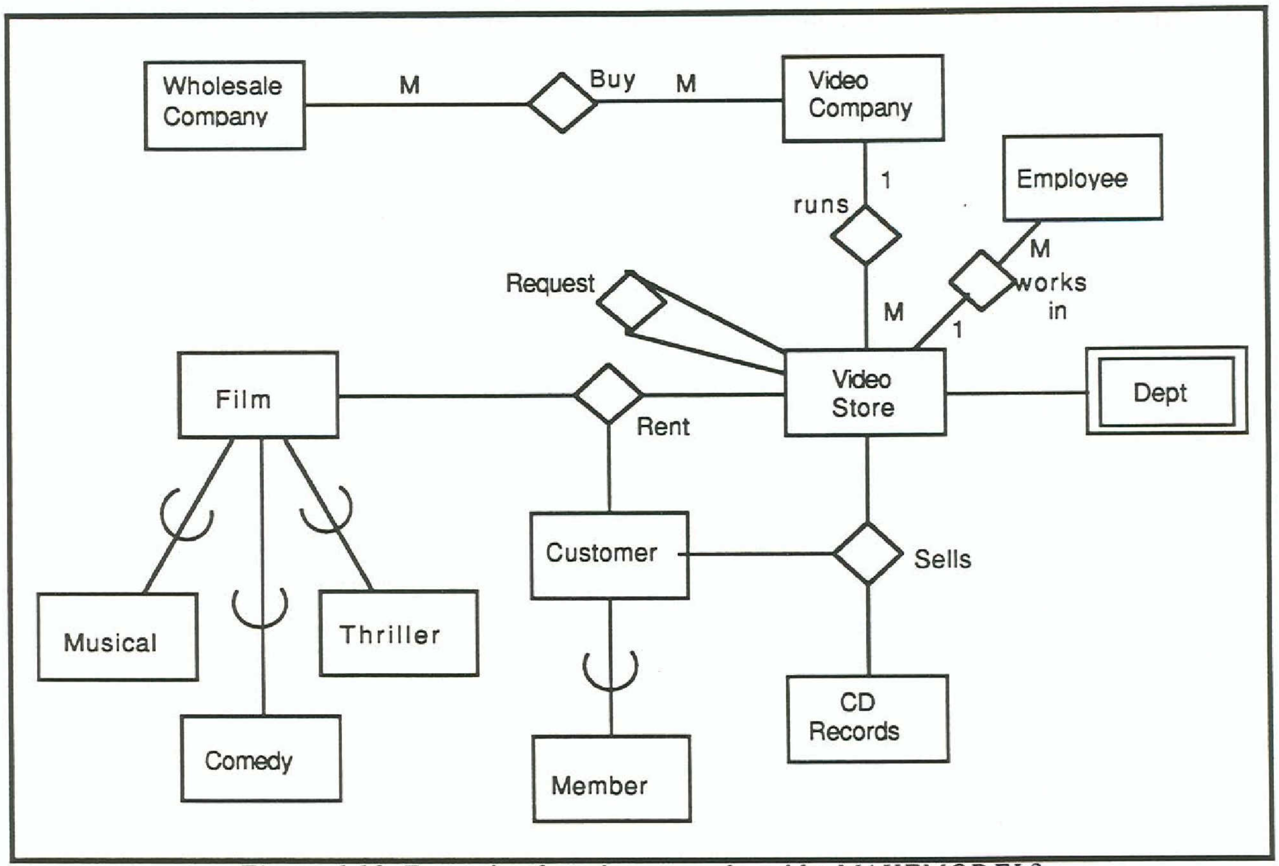


Figure 6.10 Example of a schema produced by MAKEMODEL3.

6.4 CONCLUDING EVALUATION

A new, highly automated, method for translating a relational database to a conceptual schema has been presented here. This method has been implemented in the system MAKEMODEL3. Inclusion dependencies between the different relations of the database are generated and suggested for the user. These are utilized for creating the entities and relationships of the conceptual schema and for connecting these data structures to each other. A dictionary of synonyms is used in the process of generating possible inclusion dependencies. The dictionary increases in size with each occasion the system is used for creating a new conceptual schema. This renders the system dynamic and enables it to function as a learning system.

The system is also fairly flexible as it accepts a database both with or without instances. When instances are provided, the set of potential inclusion dependencies arrived at by the system is a complete set. In this case, the dictionary of synonyms is not needed although it does become updated with information from the accepted inclusion dependencies.

The method on which MAKEMODEL3 is based appears to work on many examples of relational databases. The correctness of the method has, however, not formally been proved. This is a major project which ought to be taken up in future research.

A logical next step, to be researched, for improving the method could be an enhancement of the dictionary. One idea is to have several dictionaries, a different one for each domain. This would ensure that the dictionary remained within a reasonable size thus preventing unfeasible search time through a very large dictionary.

7

CONCLUDING REMARKS AND RECOMMENDED FURTHER RESEARCH

7.1 INCLUSION OF BACKGROUND KNOWLEDGE

While the combined method described above is more coherent than both the method based on similarity in attribute names and that based on supplementing the database with inclusion dependencies, it still falls short in the aspect of mechanization. Reading the instances of the entire database in an attempt to establish inclusion dependencies can be greatly inefficient. Moreover the necessity of user interaction, to confirm ambiguous subset/set relationships, arises far too often. Using a dictionary of synonyms does, of course, alleviate some of the extra processing, but the information required from the user is still extensive. To solve this problem, an approach which takes domain information into consideration could be examined.

Many researchers in the field of AI stress the importance of background knowledge for any agent (or process) that tries to extract semantic information from some given situation. John Sowa, in [Sowa84], proposes several ways of making such background information available. Among these are scripts and memory organization packets which are instances of schemata and prototypes. A script for a concept is a description of a sequence of events in a particular context. Memory organization packets (MOP) are like small scripts that link to other MOPs. A schema is a semantic description of a concept and a prototype is a description of a typical instance of a concept. By augmenting a database with background information, the mapping algorithm is provided with semantic information relevant to the domain of the database. This gives the algorithm a possibility of obtaining a fair amount of the information it needs to carry out the mapping from a source other than the user.

A method for mapping a relational database, supplemented with domain information, to a conceptual model, rich in semantics, is suggested for further research and a rough outline is presented below.

The basic idea is the following: two databases are supplied, a relational database (RDB) and a background information database (BIDB), which is initially empty. An attempt is made to match the relation names in the RDB to concept names in the BIDB. When no match is found, the user is asked to specify the full name of the relation under examination (the reason for this is that names may be abbreviated or slightly altered in the RDB). If a match to the full name is still not found, it means that the BIDB lacks this particular concept. The user is asked to provide information about the concept. This information is then added to the BIDB. When a match *is* found, for a concept which will map to an entity type or a subclass entity type, the appropriate entity type is created. When a match is found, for a concept which will map to a relationship type, the keys of the other relations in the RDB are examined in order to find the entities belonging to the relationship. Concepts representing relationship types, in the BIDB, are furnished with semantic information specifying the case structure of the concept. Relationships are then created from the relation specifying the relationship, relations specifying the entities belonging to the relationship and case grammar information describing the relationship.

The data describing background information can be structured in a number of different ways, any one of which could be chosen to be used here. Scripts or memory organization packets, discussed above, are two possible representations. Frames, data objects consisting of a collection of slots that describe different aspects of the objects, presented by Minsky in [Minsky75] could be used to describe entities. An enhanced predicate logic representation APC (Annotated Predicate Calculus) presented by Stepp and Michalski in [Stepp86] is a typed predicate calculus with additional operators used to describe objects and general problem-specific background knowledge. A concept dictionary, structured as a semantic net, containing deep case structure of the concepts with default values for the cases, presented by Atlerman in [Alterman85] seems to be another feasible possibility. The choice of type of data representation is an important one which merits more research.

The quality of the mapping process depends on the state of the BIDB. As the BIDB increases in size and available domain knowledge becomes more comprehensive, the need for user input decreases and the mapping process becomes more mechanized. A significant advantage of the system is that it improves each time it is used, as the BIDB is, by necessity, updated to hold more background information pertinent to the domain.

7.2 SEMANTICALLY RICHER SCHEMA

The conceptual schema created, can be made into one richer in semantic information than ER or Extended ER models. By combining the conceptual schema from the area of databases with some type of semantic network from the area of knowledge representation within artificial intelligence, an improved schema can be obtained. Perhaps one of the most expressive, best refined conceptual graph is that proposed by Sowa in [Sowa84].

An example of such a schema, suggested by Sowa, is illustrated below.

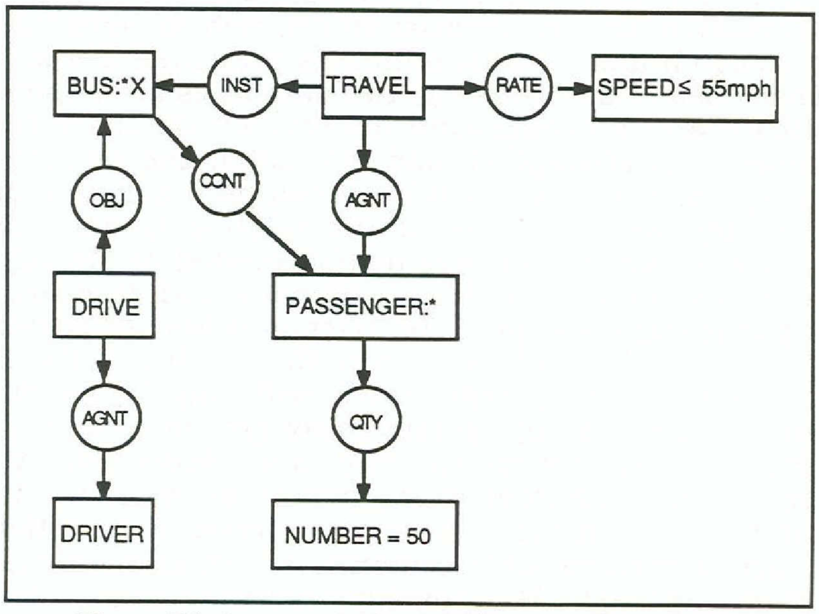


Figure 7.1 A semantically rich Conceptual graph.

The information expressed in the schema, shown in Figure 7 above, is to be read as follows: A bus contains a set of about 50 passengers, it is the instrument of travel by those passengers at a speed less than or equal to 55 miles per hour, and it is the object of driving by some driver.

This schema can, for example, be combined with the EER schema utilized by the three methods described in this paper. The case structures present in the schema could be stored in the BIDB. The presence of these structures in the BIDB would serve to facilitate the mapping process. They could also be used to make user questions, where necessary, more intelligible for the user. The most significant advantage of a combined schema lies, of course, in its capability of being able to express a greater amount of semantic information than either of the two schemas can separately.

REFERENCES

[Alterman85] R. Alterman, "A Dictionary Based on Concept Coherence", *Artificial Intelligence*, vol. 25, no. 2,1985.

[Briand84] H. Briand, "Expert System for Translating an E-R Diagram into Databases", *"Fourth Internationa Conference on Entity-Relationship Approach"*,1984

[Briand87] H. Briand, "From Minimal Cover to Entity-Relationship Diagram", *"Seventh International Conference on Entity-Relationship Approach"*, 1987.

[Bruce89] T. A. Bruce, "DA Supply, Business Demand", pp. 54-59, December 89, 1989.

[Chen76] P. P. Chen, "The Entity Relationship Model - Towards a Unified View of Data", vol. 1, no. 1, 1976.

[Chikofsky90] E. J. Chikofsky and J. H. Cross II, "Reverse Engineering and Design Recovery: A Taxonomy",pp. 13 - 17, January 1990,

[Date86] C. J. Date, "An Introduction to Database Systems", Addison-Wesley Publishing Company, 1986.

[Davis84] A. Davis, "A Methodology for Translating a Conventional File System into an Entity-Relatinship Model", *"Fourth Internationa Conference on Entity-Relationship Approach"*, 1984.

[Davis87] K. H. Davis and A. K. Arora, "Converting a Relational Database Model into an Entity Relationship Model", *"Seventh International conference on Entity-Relationship Approach"*, 1987.

[Dumpala83] S. R. Dumpala and S. K. Arora, "Schema Traslation Using the Entity-Relationship Approach", in "*Entity-Relationship Approach to Information Modeling and Analysis*", Ed. Chen, 1983.

[ElMasri85] R. ElMasri, "The Category Concept: An Extension to the Entity-Relationship Model", *Data and Knowledge Engineering*, vol. 1, no. 1,1985.

[Johannesson89] P. Johannesson and K. Kalman, "A Method for Translating Relational Schemas into Conceptual Schemas", *"Eighth International Conference on Entity-Relationship Approach"*, 1989.

[Kalman89] K. Kalman, "Implementation and Critique of an Algorithm which Maps a Relational Database to a Conceptual Model", SYSLAB Working Paper 151, Stockholm University, 1989.

[Korth86] H. Korth and A. Silberschatz, "Database System Concepts", McGraw Hill, 1986.

[Minsky75] M. Minsky, "A Framework for Representing Knowledge", in "*The psychology of Computer Vision*", Ed. P. Winston, O-Hill, New York, 1975.

[Navathe87] S. B. Navathe and A. M. Awong, "Abstracting Relational and Hierarchical Data with a Semantic Data Model", *"Seventh International Conference on Entity-Relationship Approach"*, 1987.

[Nilsson84] Nilsson, "The Translation of a Cobol Data Structure to an Entity-Relationship Type Conceptual Schema", 1984.

[Shoval87]	E.-C. Shoval, "A System for Automatic Database Schema Design Based on the Binary-Relationship Model", *Data and Knowledge Engineering*, vol. 2, 1987.
[Sowa84]	J. F. Sowa, "Conceptual Structures: Information Processing in Mind and Machine", Addison-Wesley Publishing Company, 1984.
[Stepp86]	Stepp and Michalski, "Conceptual Clustering of Structured Objects: A Goal-Oriented Approach", *Artificial Intelligence*, vol. 28, no. 1, 1986.
[Teorey86]	T. J. Teorey, "A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model", vol. 2,1986.
[Winans90]	J. Winans and K. H. Davis, "Software Reverse Engineering from a Currently Existing IMS Database to an Entity-Relationship Model", *"Entity-Relationship Approach"*, Ed. H. Kangassalo, pp. 345-360, Lausanne, Switzerland, 1990.