Semantic Dependencies in Data Modelling and Database Reverse Engineering

Tok Wang LING       Mong Li LEE
Department of Information Systems & Computer Science
National University of Singapore
{lingtw, leeml}@iscs.nus.sg

Abstract

Functional, multivalued and inclusion dependencies are only constraints for enforcing database integrity and may not imply a semantic relationship between two sets of attributes. We introduce the concept of semantic dependency to capture semantic relationship between two sets of attributes. We also introduce the notion of semantic keys. These semantic features are first identified in the Entity-Relationship approach. We illustrate the significance of semantic dependencies in data modelling and database reverse engineering.

1 Introduction

Ever since the Entity-Relationship (ER) approach was introduced by Chen [9], it attracted considerable attention in systems modelling and database design [10, 7]. The ER concepts (entities and relationships) correspond to structures naturally occurring in information systems. This enhances the ability of designers to describe accurately database applications. The ER approach to the design of relational schemas is an object-oriented alternative to the data-oriented relational normalization.

Due to the proliferation of database applications, the integration of existing databases into a federated system is one of the major challenges in responding to enterprises’ information requirements. Such integration is best performed at the conceptual model level using the ER approach [2, 20]. The process of database reverse engineering which constructs ER conceptual schemas from the existing databases facilitates the integration of these databases. This is because the ER approach possesses the necessary semantics for defining all the integration mappings that might be desired. In this paper, we introduce the concept of semantic dependency to capture semantic relationship between two sets of attributes. We also introduce the notion of semantic keys. We will identify these semantic features in the Entity-Relationship (ER) approach in Section 2. We will illustrate the significance of semantic dependencies and semantic keys in data modelling and database reverse engineering in Sections 3 and 4 respectively.

2 Semantic Dependencies in ER Approach

We briefly give the original definition of the ER approach. The ER model incorporates the concepts of entity type and relationship set. An entity type or relationship set has attributes which represent its structural properties. An attribute can be single-valued, multivalued or composite. A minimal set of attributes of an entity type E which uniquely identifies E is called a key of E. An entity type may have more than one key and we designate one of them as the identifier of the entity type. A minimal set of identifiers of some entity types participating in a relationship set R which uniquely identifies R is called a key of R. If the existence of an entity in one entity type depends upon the existence of a specific entity in another entity type, such a relationship and entity type are called existence dependent (EX) relationship set and weak entity type. A special case of weak entity types occurs when the entities cannot be identified by the values of its own attributes, but have to be identified by its EX relationship with other entities. This EX relationship set is now called an identifier dependent (ID) relationship set. In the ER model, we can have recursive relationship sets.
and special relationship sets such as \textit{ISA, UNION, INTERSECT} etc. The structure of a database organized according to the ER model can be represented by a diagrammatic technique called an Entity-Relationship Diagram (ERD) [16].

A functional dependency, $X \rightarrow Y$, besides saying that every $X$ can be associated with a unique $Y$, also intends to specify the relationship between $X$ and $Y$ [23]. Yet, functional dependencies may not represent the presence or absence of such relationships between attributes. Functional dependencies are a constraint with no influence on the way data are structured. [1, 23] shows how normalization does not necessarily result in a schema that reflects our intuition about how information should be organized in relations. Similarly, multivalued [18] and inclusion dependencies [5] are constraints to enforce the integrity of a database.

In this section, we will show that functional and multivalued dependencies are only constraints for enforcing database integrity and may not imply a semantic relationship between two sets of attributes. The concept of semantic dependency is introduced to capture semantic relationship between two sets of attributes. We illustrate with examples that the notions of semantic dependencies and semantic keys are actually embedded in the ER approach.

**Example 1** Consider the following ER diagram where we have an entity type \textit{EMPLOYEE} with identifier EMPNO, single-valued attributes SSNO, NAME, ADDRESS, AGE and a multivalued attribute QUAL (qualification). EMPLOYEE is involved in a many-to-one relationship with another entity type DEPARTMENT. The attribute STARTDATE of the relationship set WORKFOR refers to the date on which the EMPLOYEE assumes duty with the DEPARTMENT.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{ERD.pdf}
\caption{Example EMPLOYEE-DEPT ER Diagram}
\end{figure}

EMPNO is a key in the relationship set WORKFOR. We can derive the functional dependency EMPNO $\rightarrow$ STARTDATE from the relationship set. Although STARTDATE is functionally dependent on EMPNO, we observe that STARTDATE is meaningful only when associated with EMPNO and DNAME together. We say that STARTDATE is semantically dependent on \{EMPNO, DNAME\}. This is the actual meaning of the relationship construct in the ER approach. The value of STARTDATE does not make sense with just the value of either EMPNO or DNAME only. Furthermore, the value of STARTDATE will need to be updated whenever the value of DNAME of a particular employee is updated. Therefore the functional dependency EMPNO $\rightarrow$ STARTDATE does not imply the semantic dependency, denoted $\{\text{EMPNO, DNAME}\} \rightarrow^{\text{sem}} \text{STARTDATE}$. EMPNO and DNAME together is a semantic key of the relationship WORKFOR. Note that EMPNO $\rightarrow^{\text{sem}}$ DNAME is not true although EMPNO $\rightarrow$ DNAME.

The formal definitions of semantic dependency and semantic key are given below.

**Definition 1:** A key $K$ of an entity type is called a semantic key of an entity type if its value is not modifiable or is only updated offline in the event of a major policy change. Any attribute $A$ of the entity type which is not part of the semantic key is semantically dependent on $K$. We denote this semantic dependency by $K \rightarrow^{\text{sem}} A$.

The attributes SSNO, NAME, ADDRESS, AGE of the entity type EMPLOYEE in Figure 1 are both functionally and semantically dependent on the identifier EMPNO. Although EMPNO $\rightarrow$ QUAL, QUAL is semantically dependent on EMPNO. Moreover, while EMPNO, SSNO, and \{NAME, ADDRESS, AGE\} are keys in the entity type EMPLOYEE, only EMPNO and SSNO are semantic keys of EMPLOYEE. Indeed, only either EMPNO or SSNO can be the identifier of EMPLOYEE. We will explain why in the following paragraph.

The traditional requirement for an attribute to be a key of an entity type is that it uniquely identifies the entities in the entity type. However, for an attribute to qualify as a semantic key of an entity type, it must not only uniquely determine the entities in the entity type, but its value cannot be changed except in the case of a major policy change like increasing the number of digits in an attribute PHONENO. Even then, such changes will be done offline and in a controlled environment by the database administrator. We call such changes **global replacement**. On the other hand, the value of a key like \{NAME, ADDRESS, AGE\} can be changed in some update requests issued by a user. This can cause undesirable updating anomalies when such a key is used as a foreign key in a relationship. Therefore, we propose that only semantic keys of an entity type can be designated as the identifier of an entity type. We re-define the notion of identifier in an entity type as follows:

---
Definition 2: One of the semantic keys of an entity type is designated as the identifier of the entity type.

Definition 3: Let E be a weak entity type which is ID-dependent on an owner entity type and P be a minimal set of attributes of E which can uniquely identify weak entities related to the same owner entity. P and the identifier of its owner entity type together is called a semantic key K of weak entity type E if the value of P is not modifiable or is only updated offline in the event of a major policy change. Any attribute A of E which is not part of P is semantically dependent on K. We denote this semantic dependency by $K \Rightarrow^s A$. One of the semantic keys of E is designated as the identifier of E.

Definition 4: The semantic key K of a relationship set in the ER approach consists of the identifiers of all its participating entity types. An attribute A of the relationship set is semantically dependent on the semantic key of the relationship set. We denote this semantic dependency by $K \Rightarrow^r A$.

3 Significance of Semantic Dependencies in Data Modelling

The concept of semantic dependency plays an important role in data modelling. This concept enhances the ability of a database designer to ascertain whether an attribute belongs to an entity type or a relationship set. This is because we are able to characterize more precisely the semantics of attributes as we will illustrate in the following example.

Example 2 Contrast the two ER diagrams given in Figure 2a and 2b. Both are similar except for the placement of the attribute JOINDATE. JOINDATE is an attribute of the relationship set WORKFOR in Figure 2a while in Figure 2b, JOINDATE is an attribute of the entity type EMPLOYEE.

In both cases, EMPNO is a key of EMPLOYEE and WORKFOR. EMPNO also functionally determines JOINDATE. The relational schema translation of both ER diagrams is:

$$\text{EMPLOYEE (EMPNO, NAME, SALARY, JOINDATE, DNAME)}$$

However, JOINDATE is semantically different in the two ER diagrams. Assuming that an employee can be transferred from one department to another, Figure 2a says that JOINDATE is the date where an employee joins a department in an organization. Figure 2b says that JOINDATE is the date where an employee joins the organization. In Figure 2a, NAME, AGE, SALARY are semantically dependent on EMPNO, i.e. $\text{EMPNO} \Rightarrow^s (\text{NAME, AGE, SALARY})$, while JOINDATE is semantically dependent on (EMPNO, DNAME), i.e. $\text{EMPNO, DNAME} \Rightarrow (\text{JOINDATE})$.

Figure 2a: JOINDATE is an attribute of relationship WORKFOR

Figure 2b: JOINDATE is an attribute of entity type EMPLOYEE

$\text{DNAME} \Rightarrow^s \text{JOINDATE}$. If we change the value of DNAME for a particular EMPNO in the relationship WORKFOR, we will also need to change the value of JOINDATE. However, this is not so for Figure 2b as JOINDATE is only semantically dependent on EMPNO. The concept of semantic dependencies gives a more precise characterization of the semantics of attributes. Hence, the database designer will be able to ascertain whether an attribute belongs to an entity type or a relationship set.

We observe in Example 2 that the relational schema translation of an ER diagram may result in the loss of semantics. In the next example, we will illustrate that translating an ER diagram to a set of relations followed by normalization based on functional dependencies will result in loss of semantics, in particular, the loss of the relationship concept in the ER approach. We will show that translating an ER diagram to a set of nested relations based on semantic dependencies will provide a solution to the semantic loss problem.

Example 3 The ER diagram in Figure 3 is similar to Figure 2a except for some additional attributes. QUAL (Qualification) is a multivalued attribute in the entity type EMPLOYEE while JOB DUTY is a multivalued attribute in the relationship set WORKFOR. LOC is a single-valued attribute in the entity type DEPT. The identifiers of EMPLOYEE and DEPT are EMPNO and DNAME respectively. The semantic key of WORKFOR is {EMPNO, DNAME}.

$$\text{EMPLOYEE (EMPNO, NAME, SALARY, JOINDATE, DNAME, QUAL)}$$

However, JOINDATE is semantically different in the two ER diagrams. Assuming that an employee can be transferred from one department to another, Figure 2a says that JOINDATE is the date where an employee joins a department in an organization. Figure 2b says that JOINDATE is the date where an employee joins the organization. In Figure 2a, NAME, AGE, SALARY are semantically dependent on EMPNO, i.e. $\text{EMPNO} \Rightarrow^s (\text{NAME, AGE, SALARY})$, while JOINDATE is semantically dependent on (EMPNO, DNAME), i.e. $\text{EMPNO, DNAME} \Rightarrow (\text{JOINDATE})$.

Figure 3.

From the diagram, we can derive the following functional and multivalued dependencies

$$\text{EMPNO} \Rightarrow^s \text{NAME, SALARY, JOINDATE, DEPT}$$
EMPN0 → QUAL
EMPN0 → JOBDUTY
DNAME → LOC

We also have the semantic dependencies

EMPN0 ≠ NAME, SALARY, QUAL
DNAME ≠ LOC
EMPN0, DNAME → JOINDATE, JOBDUTY

After applying an ER to relational schema translation and normalization, the ER diagram can be represented by the following set of relations:

EMPLOYEE (EMPN0, NAME, SALARY, JOINDATE, DNAME)
EMPLOYEE1 (EMPN0, QUAL)
DEPT (DNAME, LOC)
WORKFOR (EMPN0, JOBDUTY)

The above set of relations does not reflect the fact that entity types EMPLOYEE, DEPT, and attributes JOINDATE, JOBDUTY are associated together in a relationship set. We also cannot ascertain whether the attributes NAME, SALARY, QUAL, JOINDATE and JOBDUTY belong to EMPLOYEE or to WORKFOR. The relation WORKFOR does not enforce the fact that JOBDUTY is semantically dependent on EMPNO and DNAME. In fact, relations EMPLOYEE1 and WORKFOR are both all-key relations. Therefore, if we design a database using the ER diagram and later translate it to a set of relations, we should maintain not only the functional dependencies, but also the semantic dependencies in the relational database. In this way, the relational database is semantically enriched for any subsequent relational to ER schema translation (see Section 4). In fact, a better relational translation of the above ER diagram would be

EMPLOYEE (EMPN0, NAME, SALARY) EMPNO
EMPLOYEE1 (EMPN0, QUAL) EMPNO
DEPT (DNAME, LOC) DNAME
WORKFOR (EMPN0, DNAME, JOINDATE) EMPNO, DNAME
WORKFOR (EMPN0, JOBDUTY) EMPNO, DNAME

The column on the right hand side shows the semantic keys of the relations. Note that the semantic key of relation WORKFOR is the same as the semantic key of WORKFOR.

We realize that translating an ER diagram to the following set of nested relations will not result in the loss of semantic information.

EMPLOYEE (EMPN0, NAME, SALARY, (QUAL)*)
DEPT (DNAME, LOC)
WORKFOR (EMPN0, DNAME, JOINDATE, (JOBDUTY)*)

where (QUAL)* and (JOBDUTY)* are nested relations (in this case, multivalued attributes) in EMPLOYEE and WORKFOR respectively. Each entity type and relationship set in an ER diagram is translated to a nested relation. The semantic dependencies are intact and there is no ambiguity as to the ownership of an attribute. It is obvious that NAME and SALARY are single-valued attributes while QUAL is a multivalued attribute in the relation EMPLOYEE. Similarly, JOINDATE is a single-valued attribute while JOBDUTY is a multivalued attribute in the relation WORKFOR. Note that we need not maintain the information of semantic keys in the nested relations. This is because we can deduce that a nested relation corresponds to a relationship if it has more than one foreign keys. For example, WORKFOR corresponds to a relationship as it contains two foreign keys, namely EMPNO and DNAME. (EMPN0, DNAME) is a semantic key of WORKFOR.

Semantic dependencies are different from functional dependencies and multivalued dependencies. Functional and multivalued dependencies are only constraints for enforcing database integrity. On the other hand, semantic dependencies capture the relationship between two sets of attributes. Therefore, semantic dependencies do not change over time. But functional dependencies may be violated or change over time because of some policy changes by the database owner. For example, if each employee (EMPN0) in a company database can only have one phone (PHONE), then we have EMPNO → PHONE. Later the company management may decide that its employees can have more than one phone, in which case EMPNO → PHONE holds instead. However, EMPNO → PHONE holds in both cases; it is not affected by such policy change.

4 Database Reverse Engineering Using Semantic Dependencies

While the concepts of semantic dependencies and semantic keys are well-defined in an ER diagram, obtaining such semantics from a relational schema is not a trivial task. This is due to an essential semantic deficiency of the relational data model compared to the ER model. In this section, we will show that the semantic dependencies has a practical application in database reverse engineering, in particular, relational to ER schema translation.

Various studies on database reverse engineering have appeared in the literature.[13] provides a general framework for database reverse engineering and heuristics for recovering a conceptual schema from a database. The transformation of hierarchical and network/CODASYL schemas has been studied by [3, 20, 24]. The transformation of relational schemas has received a great deal of attention [8, 11, 12, 14, 20, 21]. Common to these studies is the target conceptual model, which is the ER model or one of its variants.

Navathe and Awong [20] present an algorithm to convert a relational schema to an Entity-Category-
Relationship (ECR) schema using keys (primary and candidate). This causes several anomalies and limitations [15]. Johannesson & Kalman [14] extend this work by taking inclusion dependencies into account and converting to an Extended ER (EER) [22] model. Chiang's method [8] for transforming a relational to the EER schema is based on both [20] and [14]. These transformation methods do not generate a complete or accurate ER conceptual schema from the relational schema [17]. Davis & Arora [11] elaborated on the work in [4] and developed a procedure to translate a relational schema to an ER one. However, their work is not detailed enough. Entity fragmentation caused by multivalued dependency is not considered, neither is ISA relationship identified. Markowitz & Makowsky [19] present a method to convert from relational schemas to an EER model, based on key dependencies and key-based inclusion dependencies. Although their approach is formal and consistent, they did not identify important ER structures such as multivalued attributes and recursive relationship sets.

To acquire the necessary knowledge for upgrading the semantic level of the relational schema, most of these previous works either rely entirely on the DBA to obtain all the necessary knowledge interactively [20], or assume that it has been specified in advance [14, 19]. In contrast, we semi-automate the knowledge acquisition process where the relational schema is analyzed to identify implicit semantics. Our approach uses semantic dependencies in addition to inclusion dependencies, functional dependencies and multivalued dependencies in order to obtain a more complete and accurate relational to ER schema translation as compared to previous approaches.

In the relational to ER schema translation, semantic enrichment [6] is necessary to identify implicit semantics from a relational database schema. Missing semantics that could not be represented in a relational schema could be identified and made explicit in the ER diagram. Research in semantic enrichment has been concentrated on interpreting the semantics of inclusion dependencies [6, 14, 19, 25]. These semantic interpretation of inclusion dependencies is affected by the simplistic assumption that the relational key is the value-based counterpart of the concept "identifier" in semantic data models. Only entity types, relationship sets and ISA relationships are identified. Entity or relationship fragmentation caused by multivalued dependencies is not considered or may be translated wrongly in these approaches. ISA relationships are also not differentiated from one-to-one relationships. We assert that using inclusion dependencies alone may result in inaccurate translation or a proliferation of entity types. This is because inclusion dependencies does not contain semantic information. However, using semantic dependencies in addition to inclusion dependencies as in our approach, we will obtain the correct semantics and hence a more accurate translation.

It is commonly accepted that the real world consists of entities that possess properties and are connected together in some associations. To obtain a more accurate relational to ER schema translation, we want to identify these semantic concepts in the relational schema.

We formalize the concept of semantic dependencies in the relational model and develop an algorithm to semiautomatically discover the semantic dependencies in a relational schema. Further details can be found in [17].

Basically, we define the notion of entity keys. Entity keys are attributes in a relational schema which identify entities in the real world. Attributes which are not part of an entity key are property attributes. A property attribute A in a relational schema is said to be semantically dependent on entity keys \( K_1, K_2, ..., K_n \), denoted by \( K_1, K_2, ..., K_n \rightarrow A \) if

**Case 1:** \( n = 1 \)

We have \( K_1 \rightarrow A \) which states that A is a property of the entity type identified by \( K_1 \).

**Case 2:** \( n > 1 \)

The value of A needs to be updated whenever any of the value of \( K_i \), \( 1 \leq i \leq n \), is updated. Moreover, the set of \( \{K_i\} \) is maximal and occurs either in the same relation as A, say relation R, or in some other relations.

The following example illustrates how we can obtain the semantic dependencies from a database specification.

**Example 4** Suppose we have the following requirements for a company personnel database:

1. The company keeps track of each employee's name (NAME), salary (SALARY), qualifications (QUAL), and his car's registration number (REGNO). Each car can only belong to one employee. The company assigns a unique employee number (EMPNO) for each employee.
2. Each department is described by a unique name (DNAME) and a location (LOC).
3. Each employee will join a department in the company. The employees can also request to be transferred from one department to another. The company keeps track of the join date (JOINDATE) where an employee joins a department and a list of job duties (JOBDUTY) of the employee in the department.

We can obtain the following functional dependencies and multivalued dependencies which hold among the attributes from the above specification:

\[ \text{EMPNO} \rightarrow \text{NAME}, \text{SALARY}, \text{JOINDATE}, \text{DNAME} \]
\[ \text{DNAME} \rightarrow \text{LOC} \]
\[ \text{REGNO} \rightarrow \text{EMPNO} \]
EMPNO $\rightarrow$ QUAL

EMPNO $\rightarrow$ JOB DUTY

We can also obtain the semantic dependencies that hold among the attributes from the specification. It is obvious that EMPNO and DNAME are entity keys as they identify entity types Employee and Department respectively. The attributes NAME, SALARY, QUAL, and REGNO are semantically dependent on EMPNO as they are properties of the entity type Employee identified by EMPNO. Similarly, the attribute LOC is semantically dependent on DNAME. The entity types Employee and Department are involved in some association as an employee must work in some department in the company. Attributes JOINDATE and JOB DUTY are semantically dependent on (EMPNO, DNAME) since the values of these attributes may need to be updated whenever an employee is transferred to another department in the company. Therefore we have the following semantic dependencies in the above company database:

EMPNO $\rightarrow^s$ NAME, SALARY, QUAL, REGNO
DNAME $\rightarrow^s$ LOC
EMPNO, DNAME $\rightarrow^p$ JOINDATE, JOB DUTY

In logical database design, a universal relation schema is normalized based on the functional and multivalued dependencies to yield a set of 3NF or 4NF relation schemas. To reverse engineer such a relational schema, we need to identify the semantic dependencies that hold in the set of normalized relations. The ways in which semantic dependencies can occur in a set of normalized relations has been formally categorized [17]. Here, we will illustrate how semantic dependencies can occur in a relational schema with the following example.

**Example 5** Consider the following relational schema obtained for the company database in Example 4.

EMPLOYEE (EMPNO, NAME, SALARY, JOIN DATE, DNAME)
DEPT (DNAME, LOC)
EMPQUAL (EMPNO, QUAL)
EMP DUTY (EMPNO, JOB DUTY)
EMPCAR (REGNO, EMPNO)

EMPNO and DNAME are entity keys in the above relational schema. In the relation EMPLOYEE, the attributes NAME and SALARY are semantically dependent on the primary key EMPNO. Assuming that JOIN DATE is the date where an employee joins a department in a company, we have JOIN DATE semantically dependent on {EMPNO, DNAME}, a superset of the primary key of EMPLOYEE. Both EMPQUAL and EMP DUTY are all-key relations. In EMPQUAL, we have QUAL is semantically dependent on EMPNO, a subset of the primary key of EMPQUAL. Assuming that JOB DUTY in the all-key relation EMP DUTY is the list of job duties for an employee in a department, we have JOB DUTY semantically dependent on {EMPNO, DNAME}, and both EMPNO and DNAME are entity keys from relation EMPLOYEE. The entity key DNAME in the semantic dependency {EMPNO, DNAME} $\rightarrow^s$ JOB DUTY is not in the relation EMP DUTY. Furthermore, although the relations EMPQUAL and EMP DUTY are structurally similar, the semantic dependencies in these two relations are vastly different. Finally, in the relation EMP CAR, the primary key REGNO which is not an entity key, is semantically dependent on the entity key EMPNO. Therefore we have the following semantic dependencies in the above relation schemas. We qualify each semantic dependency by the relation in which it is valid.

EMPLOYEE [EMPNO $\rightarrow^s$ NAME, SALARY]
EMPLOYEE [EMPNO, DNAME $\rightarrow^p$ JOINDATE]
EMPQUAL [EMPNO $\rightarrow^s$ QUAL]
DEPT [DNAME $\rightarrow^s$ LOC]
EMP DUTY [EMPNO, DNAME $\rightarrow^p$ JOB DUTY]
EMPCAR [EMPNO $\rightarrow^s$ REGNO]

**Definition 5** Two sets of entity keys X and Y are semantically equivalent, denoted by $X \equiv Y$, if we have $X \rightarrow^s Z$ and $Y \rightarrow^s Z$ for some property attribute Z.

Semantic equivalence of two entity keys implies that they both identify the same entity type.

**Definition 6** A set of entity keys K is a semantic key of a relation R if any semantic dependency $K \rightarrow^s A$ in R implies $K \rightarrow^p K'$.

Not all relations have a semantic key. The relation EMPLOYEE in Example 5 has no semantic key. EMPNO is not a semantic key of EMPLOYEE because we have the semantic dependency {EMPNO, DNAME} $\rightarrow^s$ JOIN DATE but EMPNO and {EMPNO, DNAME} are not semantically equivalent. Similarly, {EMPNO, DNAME} is also not a semantic key of EMPLOYEE. On the other hand, EMPNO and DNAME are semantic keys of relations EMPQUAL and DEPT respectively. EMPNO is the semantic key of relation EMPCAR. We also note that the semantic key of a relation need not be entirely contained in the relation. For example, {EMPNO, DNAME} is a semantic key of the relation EMP DUTY in Example 5 although the attribute DNAME is not in the relation.

**Definition 7** A 3NF relation R in a database schema is in semantic normal form (semantic NF) if

1. R has a semantic key, and
2. any attribute of R which is not part of any semantic key of R is semantically dependent on each semantic key of R.

Relations which are not in semantic NF can be split into two or more relations which are in semantic NF.
When all the relations in a relational schema are in semantic NF, we can classify these relations into two types. The first type are relations whose semantic keys consist of only a single entity key, called entity relations. The second type are relations whose semantic keys consist of more than one entity keys, called relationship relations. In the relational to ER translation process, entity relations with the same or semantically equivalent semantic keys are translated into an entity type, while relationship relations with the same semantic keys are translated into a relationship set.

[14, 19] only use inclusion dependencies to identify entity types, relationship sets and ISA relationships. Using inclusion dependencies alone may result in inaccurate translation or a proliferation of entity types. ISA relationships are not differentiated from one-to-one relationships. Entity or relationship fragmentation caused by multivalued dependency is not considered. For example, the relation EMPDUTY is the result of fragmenting a relationship set due to a multivalued dependency EMPNO \rightarrow \rightarrow JOB DUTY. However, [14, 19] will translate this relation into a multivalued attribute of the EMP entity type or into an entity type with an ISA or many-to-many relationship connecting it to the EMP entity type. Semantically, we see that both these translations are incorrect.

The following example illustrates how we can obtain an ER schema from a relational schema using our concept of semantic dependency. It also shows what happens if we take into account only the inclusion dependencies in the translation.

**Example 6**  Given the following relational schema

\[
\begin{align*}
\text{EMPLOYEE} & \quad (\text{EMPNO}, \text{NAME}, \text{ADDRESS}) \\
\text{MANAGE} & \quad (\text{EMPNO}, \text{DNNAME}, \text{ALLOWANCE}, \text{STARTDATE}) \\
\text{DEPT} & \quad (\text{DNNAME}, \text{LOC})
\end{align*}
\]

with inclusion dependencies

\[
\begin{align*}
\text{MANAGE}[\text{EMPNO}] & \subseteq \text{EMPLOYEE}[\text{EMPNO}] \\
\text{MANAGE}[\text{DNNAME}] & \subseteq \text{DEPT}[\text{DNNAME}] \\
\text{DEPT}[\text{DNNAME}] & \subseteq \text{MANAGE}[\text{DNNAME}]
\end{align*}
\]

and semantic dependencies

\[
\begin{align*}
\text{EMPNO} \leftrightarrow \text{NAME}, \text{ADDRESS} \\
\text{EMPNO} \leftrightarrow \text{ALLOWANCE} \\
\text{EMPNO, DNNAME} \leftrightarrow \text{STARTDATE} \\
\text{DNNAME} \leftrightarrow \text{LOC}
\end{align*}
\]

\text{EMPNO} \leftrightarrow \text{ALLOWANCE} holds because we assume that the allowance of a manager is independent of the department he manages. The relation MANAGE is therefore not in semantic NF because it does not have any semantic keys. However, if we split the MANAGE relation into MANAGE1 and MANAGE2 as follows:

\[
\begin{align*}
\text{MANAGE1}[\text{EMPNO, ALLOWANCE}] \\
\text{MANAGE2}[\text{EMPNO, DNNAME, STARTDATE}]
\end{align*}
\]

Then both MANAGE1 and MANAGE2 are in semantic NF as well as in 3NF (also BCNF and 4NF). EMPNO and \{EMPNO, DNNAME\} are the semantic keys of MANAGE1 and MANAGE2 respectively. We need to generate the equivalent inclusion dependencies for the new relations as follows:

\[
\begin{align*}
\text{MANAGE1}[\text{EMPNO}] & \subseteq \text{EMPLOYEE}[\text{EMPNO}] \\
\text{MANAGE1}[\text{EMPNO}] & \subseteq \text{MANAGE2}[\text{EMPNO}] \\
\text{MANAGE2}[\text{EMPNO}] & \subseteq \text{MANAGE1}[\text{EMPNO}] \\
\text{MANAGE2}[\text{DNNAME}] & \subseteq \text{DEPT}[\text{DNNAME}] \\
\text{DEPT}[\text{DNNAME}] & \subseteq \text{MANAGE2}[\text{DNNAME}]
\end{align*}
\]

We have omitted the inclusion dependency \text{MANAGE2}[\text{EMPNO}] \subseteq \text{EMPLOYEE}[\text{EMPNO}] as it is redundant and can be derived from the other inclusion dependencies. Relations \text{EMPLOYEE}, \text{MANAGE1} and \text{DEPT} are entity relations while \text{MANAGE2} is a relationship relation. The entity relations are translated into entity types while \text{MANAGE2} is translated into a one-to-one relationship set between entity types \text{MANAGE1} and \text{DEPT}.

The inclusion dependency \text{MANAGE2}[\text{DNNAME}] \subseteq \text{DEPT}[\text{DNNAME}] indicates the referential integrity constraint in a relationship set while \text{DEPT}[\text{DNNAME}] \subseteq \text{MANAGE2}[\text{DNNAME}] indicates a mandatory participation of the entity type \text{DEPT} in the relationship set \text{MANAGE2}.

We have \text{MANAGE1} ISA \text{EMPLOYEE} since their semantic keys are equivalent and \text{MANAGE1}[\text{EMPNO}] \subseteq \text{EMPLOYEE}[\text{EMPNO}]. The entity type \text{MANAGE1} in fact represents the manager of a department. Therefore, we will obtain the following ER schema

![ER diagram](image)

[6, 14] will conclude that \text{MANAGE} is a fragmentation of \text{DEPT} or vice versa from the inclusion dependencies \text{MANAGE}[\text{DNNAME}] \subseteq \text{DEPT}[\text{DNNAME}] and \text{DEPT}[\text{DNNAME}] \subseteq \text{MANAGE}[\text{DNNAME}]. They will also conclude that \text{MANAGE} ISA \text{EMPLOYEE} from the inclusion dependency \text{MANAGE}[\text{EMPNO}] \subseteq \text{EMPLOYEE}[\text{EMPNO}]. It is obvious that their translation of the above relational schema is not correct as using inclusion dependencies alone is not sufficient to capture the semantics of the relational schema.

**5 Conclusion**

In this paper, we have shown that functional, multivalued and inclusion dependencies are only constraints
for enforcing database integrity and may not imply any semantic relationship between two sets of attributes. The concept of semantic dependency is introduced to indicate a semantic relationship between two sets of attributes. We have illustrated the significance of semantic dependencies in data modelling.

We have also highlighted how semantic dependencies in relational schema can play an important role in database reverse engineering. We have shown that using semantic dependencies in the relational schema to ER translation will improve the accuracy of the translation.

We have developed a comprehensive algorithm to reverse engineer a relational schema to an ER schema. There are two steps in this process. The first step is to semantically enrich the relational schema. Our approach is to semi-automate this enrichment process of discovering semantic dependencies in a relational schema. Human intervention is required only to resolve ambiguities when they arise and to confirm the results obtained. The second step is to use the semantic dependencies to convert the relational schema to the corresponding ER schema. Details of this algorithm can be found in [17].

References