DISCOVERING AND USING SEMANTICS FOR DATABASE SCHEMAS

by

Yuan An

A thesis submitted in conformity with
the requirements for the degree
Doctor of Philosophy

Department of Computer Science
University of Toronto

Discovering and Using Semantics for Database Schemas
Yuan An
Doctor of Philosophy
Department of Computer Science
University of Toronto
2007

ABSTRACT

This dissertation studies the problem of discovering and using semantics for structured and semi-structured data, such as relational databases and XML documents. Semantics is captured in terms of mappings from a database schema to conceptual schemas/ontologies.

Data semantics lies at the heart of data integration – the problem of sharing data across disparate sources. To address this problem, database researchers have proposed a host of solutions including federated databases, data warehousing, mediator-wrapper-based data integration systems, peer-to-peer data management systems, and more recently data spaces. In the Semantic Web community, the solution to the problem of providing machine understandable data for better web-wide information retrieval and exchange is to annotate web data using formal domain ontologies. A central issue in all of these solutions is the problem of capturing the semantics of the data to be integrated.

This dissertation describes our solutions for discovering semantics for data and using the semantics to facilitate the discovery of schema mappings. First, we develop a semi-automatic tool, MAPONTO, for discovering semantics for a database schema in terms of a given conceptual model (hereafter CM). The tool takes as inputs a relational or XML database schema, a CM covering the same domain as the database, and a set of simple element correspondences from schema elements to datatype properties in the CM. It then generates a set of logical formulas that define a mapping from the schema to the CM. The key is to align the integrity constraints in the schema with the semantic constructs in the CM, guided by standard database design principles. Second, we extend MAPONTO with a semantic approach to finding schema mapping expressions. The approach leverages the semantics of schemas expressed in terms of CMs. We present experimental results demonstrating that MAPONTO saves significant human effort in discovering the semantics of database schemas and it outperforms the traditional mapping techniques for building complex schema map-

ping expressions in terms of both recall and precision. The development of MAPONTO provides a suite of practical tools for recovering semantics for database-resident data and generating improved schema mapping results for data integration.

ACKNOWLEDGMENTS

I benefited enormously from the members of the supervisory committee for my Ph.D. program. They are Prof. John Mylopoulos, Prof. Alex Borgida, Prof. Renee Miller, and Prof. Sheila McIlraith. I am deeply grateful for their supervisions, criticisms, collaborations, encouragement, patience, tolerance, supports, and academic advice to my research and the writing of this dissertation.

I am particularly grateful to my supervisor, Prof. John Mylopoulos, who has been consistently supportive and encouraging. I feel extremely fortunate that when I entered the Ph.D. program in the Department of Computer Science at the University of Toronto five and half years ago, I had John as my supervisor. The very friendly, open, helpful, and encouraging environment in the Knowledge Management Lab led by John provided me the most positive conditions for doing interesting and innovative research and pursuing a Ph.D. degree. I would say that it might not happen for me to achieve this if I did not have the opportunity to work with John.

I am greatly indebted to Prof. Alex Borgida, my co-supervisor. It was his academic advice to my research and numerous positive suggestions to my writings that led me to overcome one hurdle after another for sharpening my skills and gaining abilities. I could not remember how many times we had spent an entire morning, an entire afternoon, or an entire day in a meeting room discussing research problems and developing algorithms together. Alex was very patient in answering my emails regarding various kinds of questions ranging from asking references to checking a lousy mathematical proof. His sharp, precise, and almost always correct comments inspired me tremendously and reshaped my mind in great deal.

I am deeply grateful to Prof. Renee Miller whose seminal work on schema mapping inspired me to develop the core idea presented in this dissertation. I benefited greatly from Renee's graduate seminar course on Data Integration in the Department of Computer Science at the University of Toronto. When I just started the Ph.D. program, I had been attracted by the influential work done by Renee. During the years of study, I was always be able to seek advice and suggestions from her. I would like to thank Renee for acting as the chair of the supervisory committee. I really appreciate those positive comments and invaluable suggestions after each committee meeting.

I am also deeply grateful to Prof. Sheila McIlraith whose expertise in AI and profound understanding about knowledge representation and reasoning helped me improve my work in many aspects. I benefited greatly from her through different occasions including advisory meetings, committee meetings, and the teaching in a graduate course.

I would like to thank Prof. Enrico Franconi for being the external examiner of this dissertation. His appraisal provided many suggestions for me to improve this dissertation.

I also would like to thank my colleagues in the Knowledge Management Lab in the Department of Computer Science at the University of Toronto. I feel very fortunate to have the opportunities working with many of them.

I would like to thank the publisher, Springer-Verlag. With the kind permission of Springer Science and Business Media, I am able to include the following material in this dissertation and allow the National Library to make use of the dissertation. The material to be reprinted is the Section 5 and 6 of the paper entitled "Discovering the Semantics of Relational Tables through Mappings", Journal on Data Semantics VII, pages 1-32, LNCS 4244, 2006 Autumn, Springer-Verlag. This material appears in Chapter 4 of this dissertation.

I would like to thank my parents for getting me started and their continued support. I am particularly grateful to my wife, Yina, for her endless patience, boundless support, and tolerating the long journey in achieving the goal of finishing this dissertation and earning the Ph.D. degree.

Finally, to my son, Jacob Yuchen An, a precious treasure from God, this dissertation is dedicated. I hope that he will soon be reading this.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This dissertation studies the problem of discovering and using semantics for structured and semi-structure data, such as relational databases and XML documents. Data semantics is captured through mappings from a database schema to domain ontologies/conceptual schemas. In this dissertation, we will use the term "conceptual model" (abbreviated as CM) to refer to a domain ontology or a conceptual schema.

1.1 Motivation

To address data integration – the problem of sharing data across disparate sources – database researchers have proposed a host of solutions including federated databases [CE87, SL90], data warehousing [Kim96, BE97, CD97], mediator-wrapper-based data integration systems [Wie92a, Ull00], peer-to-peer data management systems [BGK$^+$02, HIST03], and more recently data spaces [FHM05, HFM06]. A key component of any of these solutions is the definition of mappings between different data sources which are often heterogeneous and distributed. These *mappings* resolve the structural and semantic heterogeneity between sources and enable information sharing [Hal05]. Since large amounts of data reside in structured and semi-structured databases, such as relational tables and XML documents, building mappings between database schemas has also become a very active research area in the database community in the past two decades [BLN86, LNE89,

SP94, MHH00, NM01a, DDH01, MBR01, PVM+02, MGMR02, DR02, BSZ03, DLD+04, Kol05, MBDH05]. Despite these efforts, building schema mappings remains a very difficult problem. The reason is that building schema mappings requires understanding the semantics of schemas. The semantics of a schema specifies what objects and relationships in the subject matter are denoted by the symbols and structures in the schema. Understanding the semantics of schemas is only *approximated* by looking at linguistic, structural, and statistical evidence in schemas and the underlying data. For computerized mapping tools, the evidence is often ambiguous and insufficient for discovering the expected mappings.

In this dissertation, we employ conceptual models (CMs) such as ontologies and conceptual schemas to capture semantics for database schemas, and then use this semantics to improve traditional schema mapping tools. Capturing and using the semantics of data [Woo75, She95, BM04] is a long-standing problem in the database community. In the early days, various semantic data models were proposed to capture more "world knowledge" than the relational model, after the relational model was adopted as the main structure for managing data. Due to performance reasons, semantic data models did not prevail in building database management systems (DBMS); instead they found a place in the database design process. Databases were often first described in terms of some kind of semantic data model, e.g., Entity-Relationship model, and then converted into the syntactic structures manipulated by the underlying DBMS. The semantics of data in a database system was distributed into its operational environment, i.e., its database administrator and its application programs. This had worked well for a closed and relatively stable operational environment. For integrating and exchanging data in an ever-more distributed, dynamic, and open environment, legacy data that is inherent in this practice becomes the major obstacle since the semantics of legacy data are often inaccessible, hindering the creation of mappings between different databases.

Recently, the Semantic Web was proposed for improving information gathering and integration on the Web [BLHL01]. Data on the Semantic Web promises to be machine-understandable by being attached through semantic annotations. These annotations can be based on formal ontologies with, hopefully, widely accepted vocabulary and definitions. Similarly, we believe that annotating database-resident data with ontologies or conceptual schemas will also improve data integration and

exchange across disparate databases. This dissertation will consider a tool for annotating (capturing semantics for) relational and XML databases, and an approach for using the annotation (semantics) to improve traditional schema mapping tools.

For our purposes, a conceptual schema is normally designed for a particular application, while an ontology is intended to describe the kinds of concepts that exist in a domain, without particular applications in mind. Despite this difference, both are capable of defining semantics of a domain. Focusing on their commonalities for expressing semantics, we use a generic conceptual modeling language (CML) with features common to the Unified Modeling Language (UML), the Extended Entity-Relationship model (EER), and the Web Ontology Language (OWL), for describing a CM.

In order to capture semantics for a database schema with a CM, we need to create mappings between the database schema and the CM. These mappings can be defined at design time, when a designer transforms a conceptual model into a database schema (*top-down* approach). Alternatively, a *bottom-up* approach recovers the mapping from a database to a CM, even when they were developed independently. The *top-down* approach leads to a new, yet-to-be-defined, database design methodology. In this dissertation, we study the *bottom-up* approach for recovering the semantics of legacy data.

Formally, we use formulas of the form:

$$T(\overline{X}) \rightarrow \Phi(\overline{X}, \overline{Y}) \tag{1.1}$$

to represent the mapping from a database schema to a CM, where $T(\overline{X})$ is a formula denoting a basic organizational unit, e.g., a relational table, in the database schema, and $\Phi(\overline{X}, \overline{Y})$ is a conjunctive formula over the predicates representing the concepts, relationships, and datatype properties/attributes in the CM. As usual, variables on the left-hand side (LHS) of the implication ("$\rightarrow$") are universally quantified, and variables on the right-hand side (RHS) which do not appear on the LHS are existentially quantified. Such formulas have been used in both the Information Manifold data integration system [LSK96] and the DWQ data warehousing system [CGL$^+$01a] to define a mapping from a relational data source to a CM expressed in terms of a Description Logic. The

following example illustrates the use of this kind of a mapping formula for expressing a plausible semantics for a relational table.

Example 1.1.1. The following mapping formula might describe the semantics of the relational table student(snum, sname, dept) in a relational database schema:

$$\forall snum, sname, dept.$$
$$\text{(student}(snum, sname, dept) \quad \rightarrow \quad \exists x_1, x_2.\ \text{Student}(x_1) \wedge \text{hasNumber}(x_1, snum) \wedge$$
$$\text{hasName}(x_1, sname) \wedge \text{Department}(x_2) \wedge$$
$$\text{hasDeptNumber}(x_2, dept) \wedge \text{registeredIn}(x_1, x_2)).$$

In the formula, the term student($snum, sname, dept$) on the LHS of "$\rightarrow$" represents the relational table and its columns. The RHS is a conjunctive formula over predicates that define the concepts Student and Department, and the attributes and binary relationships hasNumber, hasName, hasDeptNumber and registeredIn. These predicates will come from a CM.

■

It is difficult, time-consuming, and error-prone to manually create the mapping formulas from a database schema to a CM, especially since the specifier must be familiar with both the intended semantics of the database schema and the contents of the CM. With a growing demand for information integration, it becomes essential to make the mapping discovery and definition process tool-supported. The first problem we will consider in this dissertation is to develop an automatic tool for creating mappings from database schemas to CMs.

An automatic tool for defining mappings from database schemas to CMs would greatly benefit to ontology-based information integration [WVV$^+$01], where manual creation of mappings from data sources to a globally-shared ontology was one of the major bottlenecks. Moreover, the representation of semantics for database schemas in terms of CMs provides an opportunity for improving traditional solutions for a second problem: that of schema mapping.

A schema mapping expression describes a relationship between a source and a target database. For example, the following is a schema mapping expression from a source relational database with

two tables Dept(deptNum, mgrNum, location) and Emp(eid, deptNum) to a target relational database with one table EmpLoc(eid, mgrNum, location):

$$\forall eid, deptNum, mgrNum, location.\ (\text{Emp}(eid, deptNum) \wedge \text{Dept}(deptNum, mgrNum, location) \rightarrow \text{EmpLoc}(eid, mgrNum, location)).$$

It is extremely difficult to find schema mapping expressions involving multiple relations in both the source and the target schemas. Traditional schema mapping solutions (e.g., Clio [MHH00]) take as input a source schema, a target schema, and some additional information from the user. The task then is to find an association among some elements in the source schema and an association among some elements in the target schema such that the pair of associations "conform" to the information supplied by the user. For schemas without explicit semantics, a solution has to look into the structures and constraints of the schemas for clues. These syntactic clues are often ambiguous and sometimes give unsatisfactory results. On the other hand, if semantics of schemas are available, it is possible to significantly increase the capabilities of traditional schema mapping tools. The other goal of this dissertation is to then utilize the semantics of schemas expressed in terms of CMs to produce improved results for schema mapping. The setting is depicted in Figure 1.1. Note that in order to accommodate a wide range of situations, we do not assume that schemas are inter-related at the CM level.

Note that the problem of discovering mappings from schemas to CMs is superficially similar to that of discovering mappings between database schemas. However, the goal of the later is finding queries/rules for integrating/translating/exchanging data, while mapping schemas to CMs is aimed at understanding and expressing the semantics of a schema in terms of a given CM. Nevertheless, both require paying special attentions to various semantic constructs in the schema and CM languages.

We elaborate on the goals of this dissertation in the next section of this chapter. We present the challenges and research issues for fulfilling these goals in Section 1.3. In Section 1.4, we overview the dissertation and illustrate the reasoning processes that underlie our solutions. We summarize the contributions that are made in this dissertation in Section 1.5. Finally, we outline the organization

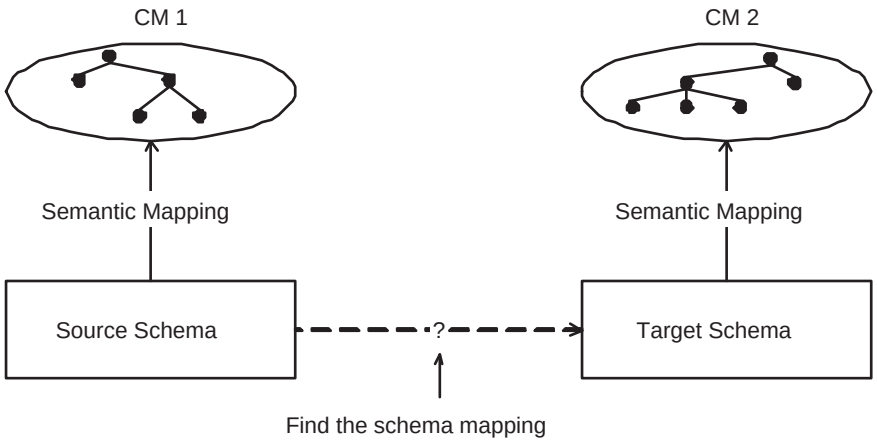

Figure 1.1: The Problem Setting of Semantic Schema Mapping

of the dissertation in Section 1.6.

1.2 Objectives

The first objective of this dissertation is to develop a tool to discover semantics for database schemas. Since we use mappings to represent semantics for database schemas, the tool is aimed at automatically discovering semantic mapping from a legacy data source to an existing CM. Given a schema and a CM, the mapping discovery process should *infer* the "correct" semantic mapping by systematically analyzing various elements in the schema and the CM. Although a fully automated tool may not be feasible, the solution should require as little human intervention as possible.

Secondly, we aim to develop a framework for using the semantics of database schemas expressed in terms of mappings to CMs. Within the framework, we want to develop a solution for discovering direct mappings between database schemas. The solution should take advantage of semantics available in schemas and in the associated CMs. Compared to traditional schema mapping techniques proposed in the literature, the solution should improve significantly the traditional techniques. To embrace broad applicability, the solution should not assume the existence of any direct connection at the semantic level.

Finally, we will implement the proposed algorithms and apply them to real world database schemas and CMs. Through experiments, our goal is to test the performance of our solutions and gain experience in building semi-automatic mapping discovery tools.

To fulfill these goals, we proceed with the following steps:

1. Identify different mapping discovery problems. Describe the input and output of each problem and the basic principles underlying proposed solutions. Set up the direction for a solution to be aimed at.

2. Develop a solution for discovering semantic mappings from relational database schemas to CMs. Test the solution using practical schemas and CMs.

3. Propose a mapping formalism for connecting XML schemas to CMs. Develop a solution for discovering semantic mappings from XML schemas to CMs. Test the solution using practical schemas and CMs.

4. Develop a solution for discovering direct database schema mappings in the presence of semantic mappings from these schemas to CMs. Test the performance of the solution by comparing it to traditional schema mapping techniques.

1.3 Challenges in Building Mappings

Using CMs to explicate semantics of logical schemas requires *matching* modeling constructs in models described in different modeling languages. In general, mapping a *model*[1] $\mathcal{S}$ to a model $\mathcal{T}$ requires deciding not only whether there is an element s in $\mathcal{S}$ that corresponds to an element t in $\mathcal{T}$, but also whether a set of elements S in $\mathcal{S}$ describes the same set of real world objects and relationships among these objects as described by a set of elements T in $\mathcal{T}$. This problem is challenging for several reasons.

First, since semantics of a legacy database have been factored out from the running system and distributed to its operational environment, to know what the data really means, one would have to

[1]Here, we use the term "model" generically to refer to either a database schema or a CM.

talk to the original creators of the database and/or check out carefully the applications that access and update the data. In most cases, however, the creators are not available and the database along with its applications has evolved dramatically. Moreover, understanding the meaning of legacy applications is a well-known intractable problem, much like the problem of data semantics.

Second, semantics is often inferred from clues in schemas and in CMs. Examples of clues in a schema are its integrity constraints and its structure. Clues in a CM include connections between concepts and constraints imposed on the connections. But the information in these clues is often incomplete. For example, a foreign key constraint in a relational database schema might represent an ordinary relationship, an ISA relationship, an identifying relationship for a weak entity, or an aggregate relationship. Each one of these has its own specific semantics.

Third, although a CM tends to describe a subject matter more directly and naturally than a database schema, the size of a CM is often large (e.g., hundreds concepts and thousands links) and relationships between concepts are complex and tangled. Given a CM, a simple and intuitive structure, e.g., a shortest spanning tree, in the CM may not be a good candidate for the semantics of a schema. (see an example in later chapter). More work on understanding semantics of schemas in terms of CMs is needed.

Fourth, even though the semantics of individual schemas in terms of CMs are available, finding the direct mapping from one schema to the other schema faces the same challenges as discovering semantics for the schemas. For example, an ISA hierarchy may be collapsed upward and represented in a single table in one relational schema, while there could be individual tables for each subclass in the hierarchy in another relational schema. Therefore, subclass tables in the latter schema need to be connected by some operators in order to be mapped to the table in the first schema. In general, there are too many ways to connect tables in a relational schema even when the semantics of individual tables are present. Worse, a mapping relationship is a pair of connections in respective schemas. If the CMs are not inter-related, it is still difficult to find matched pairs of connections at the schema level. But, hopefully, the CMs would provide more evidence for a better pairing.

Finally, mappings are often subjective, depending on the application. Therefore, users need to

be involved in the mapping process. Nevertheless, for an automatic tool, human intervention should be kept at a minimum.

1.4 Overview of the Dissertation

We now offer an overview of the dissertation and outline our solutions to the problem of discovering semantic mappings from logical schemas to CMs and the problem of discovering schema mappings between logical schemas. The fundamental problem can be described as follows: *given two models $\mathcal{S}$ and $\mathcal{T}$ about the same subject matter, for an associate $\delta_{\mathcal{S}}$ in $\mathcal{S}$, find an association $\delta_{\mathcal{T}}$ in $\mathcal{T}$ such that $\delta_{\mathcal{S}}$ and $\delta_{\mathcal{T}}$ are "semantically similar" in terms of modeling the subject matter.*

As indicated earlier, the problem is inherently difficult to automate, so interactive and semi-automatic tools are assumed to be the solution. Such a tool, e.g., Clio [MHH00], will employ a two-phase process: First, specify *simple correspondences* between "atomic elements" in the two models. There are many schema matching tools that support this phase currently (see Section 2.2). Second, derive an association among a set of elements in $\mathcal{S}$ and an association among a set of elements in $\mathcal{T}$ such that the pair of associations could give rise to a meaningful relationship between the two models. Several systems that accomplish this have been developed over the years (see Section 2.1). Up to the writing of this dissertation, Clio [MHH00] is widely considered as the best tool which quite successfully derives schema mapping formulas from a set of simple correspondences for the problem of data exchange [FKMP03].

Inspired by Clio, we will develop the solutions to our problems by also employing the two-step process: first, let the user provide a set of simple correspondences between elements in the two models being mapped, and second, derive a set of candidate mapping formulas for the user to examine. The element correspondences we consider are simple pairs of atomic elements that can be generated by most of existing schema matching tools; for example, a correspondence could be specified from a table column in a relational schema to a datatype property in a CM. In this dissertation, we focus on the second step, that is, deriving mappings from simple correspondences for the problems we consider.

A mapping between two models consists of a set of relationships between associations in the two models. An association in a model describes a connection among a set of elements in the model. Intuitively, a mapping relationship should be an equivalent relationship which means that the two associations being related are equivalent in terms of modeling a subject matter. More specifically, for two models $\mathcal{S}$ and $\mathcal{T}$ about the same subject matter, we say that there is a mapping relationship between an association $\delta_{\mathcal{S}}$ in $\mathcal{S}$ and an association $\delta_{\mathcal{T}}$ in $\mathcal{T}$ if both $\delta_{\mathcal{S}}$ and $\delta_{\mathcal{T}}$ describe the same set of objects and the same particular connection among objects in the set according to the subject matter. In regard to a mapping discovery solution, this definition directs us to find semantically equivalent associations in two models. Unfortunately, with limited information available, an automatic mapping discovery tool only can approximate the semantically equivalent associations by "semantically similar" associations. For example, if a tool recognizes that an association in model $\mathcal{S}$ describes two entities Project and Employee, and the managerOf relationship between these two entities, then it would attempt to discover an association in model $\mathcal{T}$ which also describes Project and Employee, and a "semantically similar" relationship to managerOf, such as contactPerson. The semantic similarity would be supported by the fact that both relationships are functional from Project to Employee and are **partOf** relationships. In this dissertation, we assume that the names of schema elements are merely syntactic strings indistinguishable in our solutions.

Given a set of correspondences between elements in two models, our solutions thus are to find $\delta_{\mathcal{S}}$ and $\delta_{\mathcal{T}}$ such that they are "semantically similar". In measuring the performance of our solutions, we will test them against real-world applications and use externally provided correct mappings as "gold standard". In the next two subsections, we illustrate our solutions by means of examples. In Chapter 2 entitled "Related Work" and the following chapters dedicated to the development of solutions to our problems, we present the differences between our solutions and the existing technique in Clio and explain how we advance the state of the art.

1.4.1 Discovering Semantic Mappings from Database Schemas to CMs

In discovering a semantic mapping from a database schema $\mathcal{S}$ to a CM $\mathcal{O}$, we craft our heuristics based on a careful study of standard database design process relating the constructs of the schema modeling language with those of conceptual modeling language. Suppose the schema $\mathcal{S}$ was derived from a conceptual model $\mathcal{E}$, then we can use associations in $\mathcal{O}$ to estimate the associations in $\mathcal{E}$ from which the basic organization units of $\mathcal{S}$ were derived. The following example illustrates the reasoning process for discovering a semantic mapping from a relational schema to a CM.

Example 1.4.1. Figure 1.2 shows a relational table project(num, supervisor) and the enterprise CM.

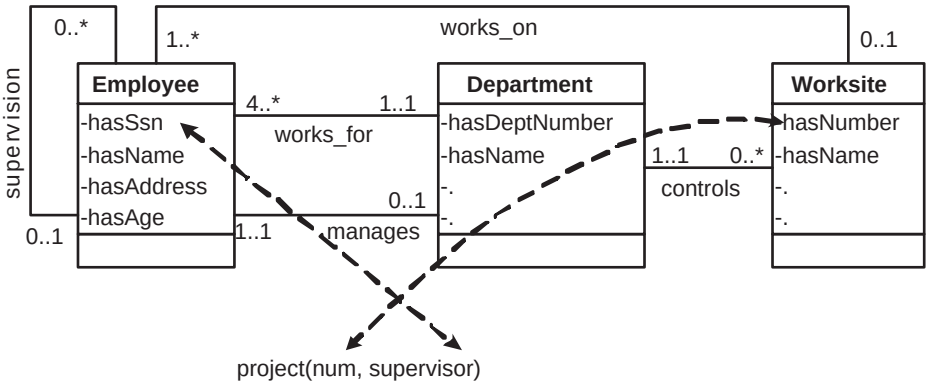

Figure 1.2: Discovering the Semantics of A Table

Suppose we wish to discover the semantics of the table project(num, supervisor) with key num in terms of the enterprise CM. Suppose that by looking at column names and the CM graph, the user draws the simple correspondences shown as dashed arrows in Figure 1.2. This indicates, for example, that the num column corresponds to the hasNumber property of the Worksite concept. Using prefixes $\mathcal{S}$ and $\mathcal{O}$ to distinguish tables in the relational schema and concepts in the CM (both of which will eventually be thought of as predicates), we present the correspondences as follows:

$$\mathcal{S}\text{:project.num} \leftrightsquigarrow \mathcal{O}\text{:Worksite.hasNumber}$$

$$\mathcal{S}\text{:project.supervisor} \leftrightsquigarrow \mathcal{O}\text{:Employee.hasSsn}$$

The information in the table indicates that for each value of hasNumber (and hence instance of Worksite), the table project associates at most one value of hasSsn (instance of Employee). Hence the association between Worksite and Employee induced by the project table should be functional from Worksite to Employee. Consequently, the association will be the manager of the department controlling the worksite, and our solution will return the composed association controls$^-$∘manages$^-$, where controls$^-$ and manages$^-$ are the inverse of controls and manages, respectively.

On the other hand, if both columns num and supervisor were key for the project table, values in column num were intended to be associated with multiple values in column supervisor, and conversely – otherwise the table would had been specified to have a smaller key. Therefore, a functional association would likely not reflect a proper semantics of the table. In this case, the association would be the workers of the department controlling the worksite, and our solution will return the composed association controls$^-$∘works_for$^-$.

At the end, the solution produces a list of plausible mapping formulas, which will include the following formula, expressing a possible semantics for the table:

$$\begin{aligned} \mathcal{S}\text{:project}(num, supervisor) \quad \rightarrow \quad & \mathcal{O}\text{:Worksite}(x_1),\ \mathcal{O}\text{:hasNumber}(x_1,num), \\ & \mathcal{O}\text{:Department}(x_2),\ \mathcal{O}\text{:Employee}(x_3), \\ & \mathcal{O}\text{:hasSsn}(x_3,supervisor),\ \mathcal{O}\text{:controls}(x_2,x_1), \\ & \mathcal{O}\text{:manages}(x_3,x_2). \end{aligned}$$

■

We have chosen to flesh out the above reasoning principles in a systematic manner by considering the behavior of our proposed solution on database schemas designed from CMs, e.g., Extended Entity Relationship (EER) diagrams. Database design methodology is a technique widely covered in undergraduate database courses. For relational schema, we refer to this as er2rel *schema design*. One benefit of this approach is that it allows us to prove that our algorithm, though heuristic in general, is in some sense "correct" for a certain class of schemas. Of course, in practice such schemas may be "denormalized" in order to improve efficiency, and only parts of the CM may be

realized in the database. Our solution uses the general principles enunciated above even in such cases, with relatively good results in practice. In Chapter 4, we focus on the problem of discovering semantics for relational schemas. We first identify that the existing technique in Clio [PVM+02] does not provide the expected results to our problem. We then proceed to develop a novel solution. In Chapter 5, we study the problem of discovering semantics for XML schemas. We differentiate the solution for this problem from that for the problem of relational case. In particular, the solution for XML schemas will make use of the parent-child hierarchical relationships which carry much semantics. In addition, the analysis of the occurrence constraints imposed on the parent-child relationships is one of the aspects that make our solution different from the Clio's technique for XML schema mapping [PVM+02], where chase on nested-referential integrity constraints lies at its core.

1.4.2 Discovering Mappings between Database Schemas

Unlike existing approaches for finding mappings between database schemas, our solution assumes the presence of the semantics of each schema, expressed in terms of a mapping to a CM. Given a relational schema $\mathcal{S}$ associated with a CM through a semantic mapping $\Sigma_{\mathcal{S}}$ and a relational schema $\mathcal{T}$ associated with a CM through a semantic mapping $\Sigma_{\mathcal{T}}$. Let L be the set of correspondences linking a set $L(\mathcal{S})$ of columns in $\mathcal{S}$ to a set $L(\mathcal{T})$ of columns in $\mathcal{T}$. To find an association $\delta_{\mathcal{S}}$ among columns in $L(\mathcal{S})$ and an association $\delta_{\mathcal{T}}$ among columns in $L(\mathcal{T})$ such that $\delta_{\mathcal{S}}$ and $\delta_{\mathcal{T}}$ are semantically similar in modeling a subject matter, we will leverage the semantics encoded in $\Sigma_{\mathcal{S}}$ and $\Sigma_{\mathcal{T}}$.

Example 1.4.2. Consider the source relational schema given on the left side of Figure 1.3. It contains a single table officeEquipment(equipID, faxNo, printerName). The semantics of the source schema is encoded by associating the table with the CM above it. On the right side, there is a target schema containing three tables: machine(serialNo), faxMachine(serialNo, faxNo), and printer(serialNo, printerName); the dashed arrows indicate referential integrity constraints over schema elements. The semantics of these tables are encoded by associating them with the target CM. Underlined column names mean that the column is part of a primary key of the table. And we

use a keyword key in a CM to indicate that the attribute with this keyword is an identifier for the entity it is associated with. Let us assume that attributes of entities are single-valued and simple.

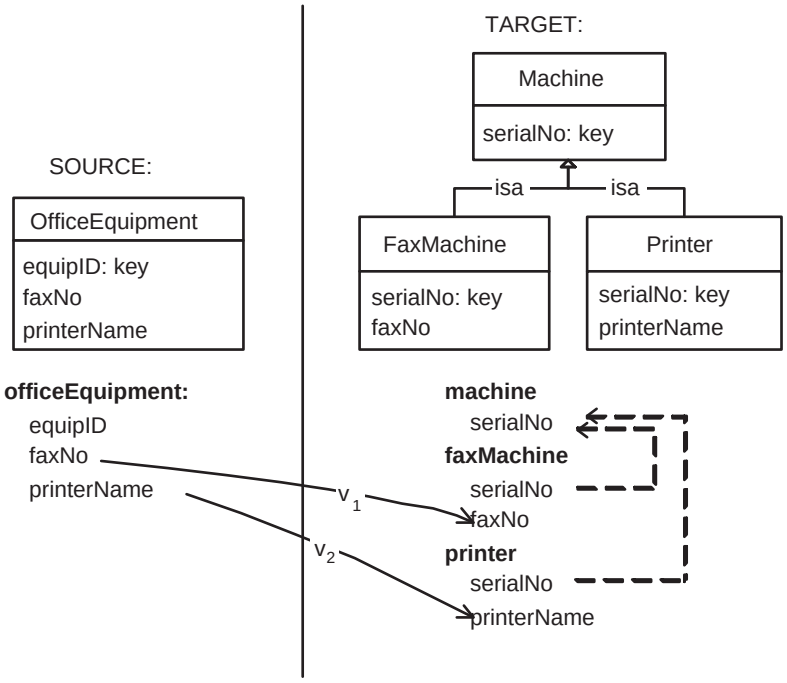

Figure 1.3: Discovering the Mapping between Schemas

To discover a mapping between the source and target schemas, the user specifies the simple element correspondences v_1 and v_2. If the semantics of the target CM indicates that the ISA hierarchy is overlapping, our solution will use the ISA links to connect the three entities and generate the following mapping formula by translating the CM connection into the formula:

$$M\text{: } \forall equipID, faxNo, printerName.(\text{officeEquipment}(equipID, faxNo, printerName) \rightarrow \text{machine}(serialNo) \wedge \text{faxMachine}(serialNo, faxNo) \wedge \text{printer}(serialNo, printerName)).$$

The solution first finds correspondences between elements in the CMs by lifting up the original correspondences at the logical schema level to the CM level. It does so by following the semantic mappings from the logical schemas to the CMs. The resulted correspondences indicate that the attribute faxNo of the entity OfficeEquipment in the source CM corresponds to the at-

tribute faxNo of the entity FaxMachine in the target CM and the attribute printerName of the entity OfficeEquipment in the source CM corresponds to the attribute printerName of the entity Printer in the target CM. Next, the solution attempts to find an association in the source CM between attributes faxNo and printerName and an association in the target CM between attributes faxNo and printerName such that the two associations are "semantically similar". In the source CM, the two attributes are associated with the single entity OfficeEquipment, while in the target faxNo and printerName are associated with entities FaxMachine and Printer, respectively. Since FaxMachine and Printer are subclasses of Machine, the ISA hierarchy could be collapsed upward so that all attributes of subclasses are associated with the superclass. This conceptual transformation would result in an "semantically equivalent" CM to the source CM. Therefore, the solution maps the single entity in the source to the ISA hierarchy in the target.

■

In contrast, a logical approach that attempts to join all of the target tables together in order to establish the mapping to the source table [PVM$^+$02] may produce too many join expressions in general. Therefore, a logical approach usually follows one direction to chain referential constraints for joining tables. As a result, the expected mapping is very likely not to be discovered (see Example 2.1.1). In Chapter 6, we show a number of scenarios where existing solutions do not discover the expected mapping expressions. We then propose a semantic approach to improved schema mapping discovery.

1.5 Contributions of the Dissertation

First, manual creation of mapping formulas expressing semantics of a database schema in terms of a CM is inefficient and ineffective. Although there are a number of (semi-)automatic tools for deriving complex mapping formulas between database schemas (e.g., Clio [MHH00, PVM$^+$02]), it is not clear whether the techniques employed by the schema mapping tools are appropriate to the problem of discovering semantics for schemas. Second, we observe that existing solutions for mapping creation often do not provide the underlying meaning of the mapping they are deriving,

and the user has to delve into the very details of the algorithms to understand the implications of the mapping derivation processes. Third, we observe that existing solutions usually do not provide any sense of "correctness" about a proposed mapping. It is the user's responsibility to decide whether a mapping solution is effective to her scenario. Fourth, since semantics of database schemas are not available, existing solutions to schema mapping often only explore the information encoded in schema constraints or carried by the data. It is not clear how to leverage the semantics encoded in the semantic mappings from schemas to CMs. Finally, it is not clear whether it is feasible and beneficial to use the semantics for deriving schema mappings. By exploring and answering the above questions, we make the following contributions in this dissertation:

- The problem of data semantics is formulated in terms of mappings and mapping discovery. We take that semantics of data sources can be explicated by setting up semantic mappings from the schemas describing data to CMs. We therefore identify a new version of data mapping problem: That of discovering mapping formulas expressing semantics of database schemas from the schemas, interpreting CMs, and simple user inputs – element correspondences.

- We describe the underlying meaning of the mapping we are looking for, and we observe that existing solutions to schema mapping discovery do not satisfy our description and the techniques employed by these solutions ignore important knowledge in data modeling. We then propose algorithms that are enhanced to take into account information about the schemas, the CMs, and standard database design guidelines.

- In discovering semantics of database schemas, we identify an important core class of problems that are often encountered in practice. We give formal results proving that the otherwise heuristic algorithm is correct for this class of problems. Thus a user can easily identify the effectiveness of the mapping solution for their scenarios.

- Another important contribution of this dissertation is that we demonstrate that the explication of semantics of database schemas benefits to the long-standing and increasingly important problem of sharing data across disparate data sources. We study the problem of discovering

schema mapping expressions using data semantics even though different data sources are not connected at the semantic level. We suggest that the semantics of database schemas should be established and maintained for managing information in an open, distributed, and dynamic environment.

- To test the effectiveness and usefulness of our solutions in practice, we implemented the algorithms in a prototype tool and applied it to a variety of data sets. We argue, for tools producing complex artifacts, it may be reasonable to measure success not just by the number of cases where the tool produced the exact answer desired, but also by the ease with which incorrect answers can be modified to produce correct ones. Our tool is successful because it can reduce the amount of human effort required to perform a given task.

1.6 Organization of the Dissertation

The content of this dissertation is organized such that each chapter is relatively self-contained. Chapters 1, 2, and 3 present the problems and the main ideas for solving these problems in this dissertation, along with relations to existing work. The remaining chapters describe detailed solutions to specific problems.

In particular, Chapter 2 contrasts our approach with related work in the literature. Chapter 3 introduces some formal notations and describes three specific mapping discovery problems. Chapter 4 studies a solution to the problem of discovering semantics for relational tables and gives evaluation results. Chapter 5 presents a mapping formalism for capturing semantics for XML schemas and develops an algorithm for finding such semantics. This chapter also includes test results of the algorithm on real data. Chapter 6 proposes a framework for using data semantics for discovering schema mapping expressions. This chapter studies a semantic solution for generating schema mappings and presents the experimental results on mapping performance in comparison with traditional techniques. Finally, Chapter 7 summarizes the entire work and points to future research directions.

Some of the results have been published in conference proceedings and journals. Specifically, the AAAI-2006 paper [AMB06] summarizes solutions for building semantic mappings from databases

to CMs. The ODBASE-2005 paper [ABM05b] and the JoDS journal paper [ABM06] detail the solution to discovering semantics of relational tables. The ISWC-2005 paper [ABM05a] presents an algorithm for discovering semantic mappings from XML schemas to CMs. Finally, the ICDE-2007 paper [ABMM07] contains results on using data semantics for discovering schema mapping expressions.

Chapter 2

Related Work

In this chapter we review work that is related to our research on discovering semantic mappings from database schemas to CMs, also on discovering schema mapping using the semantic mappings. We will discuss in relevant sections how our work advances the state of the art. We start by reviewing previous solutions for discovering mappings in Section 2.1. Most of the solutions were designed for mappings between database schemas. We show why the problem of semantic mapping in this dissertation is new, and we contrast our solution to previous techniques. Since element correspondences play a critical role in our solution as well as in many previous solutions, we survey tools for generating element correspondences in Section 2.2. Mappings are first-class citizens in the framework of model management. We review this piece of work in Section 2.3. We differentiate our work on discovering semantics for database schema from the work on data reverse engineering in Section 2.4. In Section 2.5, we briefly survey achievements made in query processing over mappings. Finally, we discuss conceptual modeling, which is related to the problem of data semantics in Section 2.6.

2.1 Schema Mapping for Data Integration and Exchange

Most approaches to discovering mappings focused on database schemas, some on ontologies. *Schema mapping* is the problem of finding a "meaningful" relationship from a *source* database schema to a

target database schema. The relationship is expressed in terms of logical formulas and the mapping is often used for data integration, exchange, and translation. The following example is adopted from the original paper of the Clio schema mapping tool [MHH00]. Figure 2.1 shows a source relational schema S containing two tables, address and engineer, and a referential integrity constraint r between the two tables. The figure also shows a target relational schema T with a single table, employee. To translate a data instance of S to a data instance of T as guided by the correspondences v_1, v_2, v_3, and v_4, which indicate, for instance, that the data objects under the addr column of the address table become the objects under the addr column of the employee table, one can use the following First Order Logic formula to represent a "meaningful" relationship between the source and the target:

$$\forall id, addr, name, sal.(\text{address}(id, addr) \wedge \text{engineer}(id, name, sal) \rightarrow \text{employee}(id, name, sal, addr)).$$

This mapping can be used to create an employee tuple by joining together an engineer tuple and an address tuple. Of course, there are other possible mapping formulas between the two schemas.

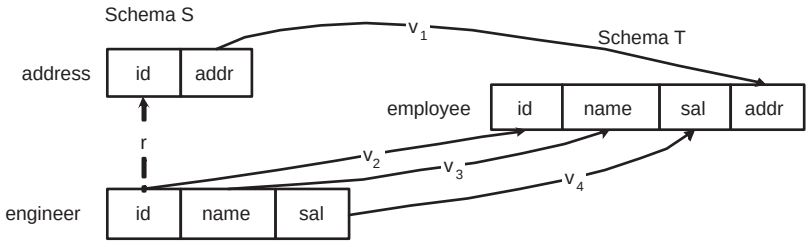

Figure 2.1: Deriving Schema Mapping Expressions for Data Exchange

Mappings are fundamental to many applications. To study them in a general, application-independent way, Madhavan et al. [MB$^+$02] proposed a general framework for mappings between domain models, where a domain model denotes a representation of a domain in a formal language, such as a relational schema in the relational formalism. Given two domain models T_1 (in a language $\mathcal{L}_1$) and T_2 (in $\mathcal{L}_2$), a mapping between T_1 and T_2 may include a helper domain model T_3

(in language $\mathcal{L}_3$), and consists of a set of formulas each of which is over (T_1, T_2), (T_1, T_3), or (T_2, T_3). A mapping formula over a pair of domain models (T_1, T_2) is of the form e_1 **op** e_2, where e_1 and e_2 are expressions over T_1 and T_2, respectively, and **op** is a well-defined operator. For example, both e_1 and e_2 can be query expressions over two relational databases T_1 and T_2. The query results of e_1 and e_2 should be compatible, and **op** can be a subset relationship between the two queries. The semantics of the mapping is given by the interpretations of all domain models involved such that these interpretations together satisfy all the mapping formulas. The authors also propose and study three properties of mappings: *query answerability*, concerning whether a mapping is adequate for answering certain queries; *mapping inference*, concerning whether a mapping is minimal; and *mapping composition*, concerning the composition of two mappings.

The primary use of mapping is for *data integration*. Data integration often combines data from disparate sources and provides users with a unified view of these data [Len02]. The typical architecture of a data integration system consists of a global schema, a set of data sources (local schemas), and mappings between the sources and the global schema. The sources provide the real data, while the global schema is an integrated and reconciled view of the real data. There are two ways for providing the global view: a materialized view [Wid95] and a virtual view [Wie92b]. In the materialized view approach, a.k.a. *data warehousing*, source data are integrated and materialized in a database under the global schema. In the virtual view approach, data remain in the sources. The global schema is connected through mappings to the source schemas so that user queries against the global view can be answered by reformulating them into queries over source databases. In both approaches, mappings between the global schema and the source schemas are the main vehicle for integration.

Formally, a data integration system is a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where $\mathcal{G}$ is the global schema in a language $\mathcal{L}_\mathcal{G}$, $\mathcal{S}$ is the source schema in a language $\mathcal{L}_\mathcal{S}$, and $\mathcal{M}$ is the mapping between $\mathcal{G}$ and $\mathcal{S}$. The mapping consists of a set of assertions of the forms $Q_\mathcal{S} \hookrightarrow Q_\mathcal{G}$ and $Q_\mathcal{G} \hookrightarrow Q_\mathcal{S}$, where $Q_\mathcal{S}$ and $Q_\mathcal{G}$ are queries of the same arity, respectively over the source schema $\mathcal{S}$ and the global schema $\mathcal{G}$. The semantics of a mapping is specified by a legal global database satisfying the mapping with respect to a source database. In particular, there are two basic approaches for specifying the

mapping: local-as-view (LAV) [LSK96] and global-as-view (GAV) [Ull00].

The mapping in the LAV approach associates to each element s of the source schema $\mathcal{S}$ a query $Q_{\mathcal{G}}$ over $\mathcal{G}$. Hence, a LAV mapping consists of a set of assertions of the form: $s \hookrightarrow Q_{\mathcal{G}}$. In the GAV approach, the mapping associates to each element g in $\mathcal{G}$ a query $Q_{\mathcal{S}}$ over $\mathcal{S}$. Therefore, a GAV mapping consists of a set of assertions of the form: $g \hookrightarrow Q_{\mathcal{S}}$. From the modeling point of view, the LAV approach is based on the idea that the content of each source should be characterized in terms of a view over the global schema, while in the GAV approach the idea is that the content of each element of the global schema should be characterized in terms of a view over the sources. In principle, the LAV approach favors extensibility, while the GAV favors query processing. Examples of LAV system are Information Manifold [LSK96] and the XML data integration system in [MFK01]. Examples of GAV system are TSIMMIS [GMPQ$^+$97] and Garlic [GHS$^+$95].

To specify the semantics of the operator "$\hookrightarrow$" in the mapping formula $Q_{\mathcal{S}} \hookrightarrow Q_{\mathcal{G}}$, three possibilities have been considered in the literature, *sound*, *complete*, and *exact*. A *sound* mapping $Q_{\mathcal{S}} \subseteq Q_{\mathcal{G}}$ means that the answers provided by the query $Q_{\mathcal{S}}$ is contained in the answers provide by the query $Q_{\mathcal{G}}$. A *complete* mapping $Q_{\mathcal{S}} \supseteq Q_{\mathcal{G}}$ means that the answers provided by the query $Q_{\mathcal{S}}$ contains the answers provide by the query $Q_{\mathcal{G}}$. Finally, an *exact* mapping means that these two set of answers are equivalent, i.e., both sound and complete.

In general, a mapping formula $Q_{\mathcal{S}} \hookrightarrow Q_{\mathcal{G}}$ relates an expression/query $Q_{\mathcal{S}}$ to an expression/query $Q_{\mathcal{G}}$. This is the so-called global-local-as-view (GLAV) approach [FLM99]. More precisely, in a GLAV mapping as introduced in [FLM99], $Q_{\mathcal{S}}$ is a conjunctive query over the source schema and $Q_{\mathcal{G}}$ is a conjunctive query over the global schema. Such a formalism is used in practical data exchange systems [FKMP03, FKP03, AL05, Kol05], where the mapping is called a *source-to-target tuple generating dependency (s-t tgd)*. In particular, a data exchange setting consists of a source schema $\mathcal{S}$, a target schema $\mathcal{T}$, a set of target dependencies, and a set of s-t tgds. Each s-t tgd is of the form [FKMP03]

$$\forall \overline{x}(\phi_{\mathcal{S}}(\overline{x}) \rightarrow \exists \overline{y} \psi_{\mathcal{T}}(\overline{x}, \overline{y})), \tag{2.1}$$

where $\phi_{\mathcal{S}}(\overline{x})$ is a conjunction of atomic formulas over $\mathcal{S}$ and $\psi_{\mathcal{T}}(\overline{x}, \overline{y})$ is a conjunction of atomic formulas over $\mathcal{T}$.

Given two database schemas – either a global schema and a source schema in the data integration setting, or a source schema and a target schema in the data exchange setting, finding the mapping between the two schemas is a difficult problem. Nevertheless, people have striven to develop tools for helping users in deriving schema mappings. Example tools are TranSem [MZ98], Clio [MHH00, PVM+02], HePToX [BCH+05], MQG [KB99], and the XML data integration system presented in [KX05]. As we said before, such a tool usually adopts a two-step paradigm: First, specify some simple correspondences between schema elements; there are several tools that support this task, and we will survey them in the next section. Then derive plausible declarative mapping expressions for users to select from. The selection process may be assisted using the actual data stored in the database [YMHF01]. The primary principle of the current solutions to deriving mappings is using integrity constraints (especially referential integrity constraints) in a schema to assemble "logically connected elements". These logical associations, together with the element correspondences, then give rise to mappings between schemas. We refer to the current solutions as *Referential-Integrity-Constraint-based* (abbreviated *RIC-based*) techniques.

The following example illustrates how the typical RIC-based technique appeared in Clio [PVM+02] derives schema mappings.

Example 2.1.1. Figure 2.2 shows a pair of relational schemas which are reproduced from the Figure 1.3.

A dashed arrow represents a referential integrity constraint (RIC), i.e., a foreign key referencing a key. As shown in the figure, there are two RICs, written textually as r_1: faxMachine.serialNo $\subseteq$ machine.serialNo and r_2: printer.serialNo $\subseteq$ machine.serialNo. To generate a declarative mapping expression, the RIC-based Clio technique [PVM+02] employs an extension of the relational chase algorithm to first assemble logically connected elements into so-called logical relations/associations. The result of chasing the table faxMachine(serialNo, faxNo) using the RIC r_1 can be represented by the following algebraic expression:

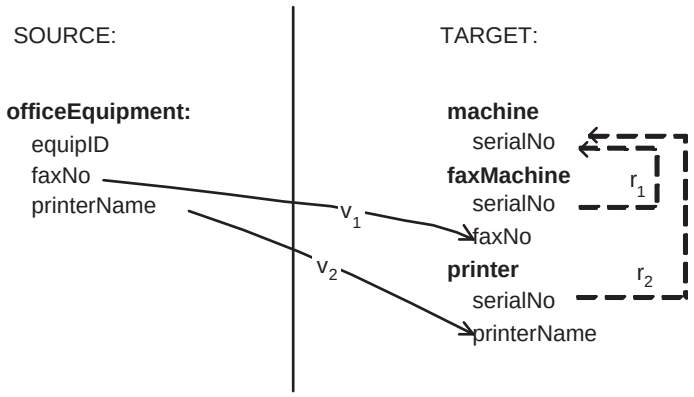

Figure 2.2: Deriving Schema Mapping by a RIC-based Technique

T_1: faxMachine(serialNo, faxNo)⋈machine(serialNo).

Since no further chase steps can be applied to T_1 (i.e., T_1 cannot be expanded further), T_1 is a logical relation. Likewise, the result of chasing the table printer(serialNo, printerName) using the RIC r_2 is the logical relation:

T_2: printer(serialNo, printerName)⋈machine(serialNo).

In the source, the logical relation is S_1: officeEquipment(equipID, faxNo, printerName).

A mapping is a pair of a source and a target logical relations such that the pair covers some correspondences specified by the user. The pair $\langle S_1, T_1 \rangle$ covers v_1 and the pair $\langle S_1, T_2 \rangle$ covers v_2. Therefore, the RIC-based technique generates the following two mapping candidates in the form of s-t tgd:

M_1: $\forall equipID, faxNo, printerName.$(officeEquipment($equipID$, $faxNo$, $printerName$)
$\rightarrow \exists serialNo.$ faxMachine($serialNo$, $faxNo$)∧machine($serialNo$)).

M_2: $\forall equipID, faxNo, printerName.$(officeEquipment($equipID$, $faxNo$, $printerName$)
$\rightarrow \exists serialNo.$ printer($serialNo$, $printerName$)∧machine($serialNo$)).

■

In principle, an RIC-based technique first looks for logical relations in each schema, and then pairs up logical relations in different schemas to cover the given correspondences. In contrast, our solution focuses on discovering a pair of "semantically similar" associations with the assistance of the semantics of schemas expressed in terms of CMs. As a result, our solution would greatly reduce the number of mapping candidates by eliminating many suspicious pairings.

Schema Integration A relevant problem to schema mapping is *schema integration* which is the problem of building a database schema by unifying a set of individual schemas, for example, creating a global schema from a set of local schemas for a data integration system. A commonality existing in both schema mapping and schema integration is resolving heterogeneity due to different schemas. The works in [KLK91], [KCGS93], and [SK92] have shed light on classifying a variety of conflicts causing heterogeneity. Batini [BLN86] presented a comprehensive survey on schema integration in the late 1980s. Methods surveyed in [BLN86] are a mixture of techniques involving exploring and resolving conflicts from naming to structures. In contrast, Spaccapietra [SP94] used the *Real World State* as the semantics of the elements and pieces of structures of schemas for integration. To measure the "relativism" of data structures, i.e., the ability to structure data in different ways, Hull [Hul84] introduced the notion of *information capacity*, and Miller [MIR93] studied the problem of information capacity preservation in schema integration.

Mapping Adaptation, Composition, and Inversion As mapping becomes a fundamental component in modern information management systems, other problems concerning managing and manipulating mappings are also attracting increasing attention. Specifically, mapping adaptation [VMP03, YP05] is the problem of maintaining the validity of mappings under schema evolution by reusing the original mappings. Mapping composition [MH03, FKPT04, NBM05] is concerned with generating a direct mapping between two data sources by composing the mappings that relate both data sources to an intermediary data source. Related to mapping composition, inverting mappings [Fag06] is aimed at constructing an inverse of a mapping. Finally, mappings are represented in the form of second-order s-t tgds [FKPT04] for mapping composition, and mappings are nested [FHH$^+$06] for grouping target instances during the actual data translation process.

2.1.1 Ontology-Based Information Integration

As ontologies have gained growing attention over the past decade, many people have used an ontology as the global schema for a data integration system, building so-called *ontology-based information integration systems*, e.g., Carnot [CHS91], SIMS [AKS96], OBSERVER [MIKS96], Information Manifold [LSK96], InfoSleuth [BBB+97], PICSEL [GLR99], and DWQ [CGL+01a]. In these systems, mappings are specified from data sources to an ontology acting as a global schema. As in the traditional data integration systems, two types of formalisms for specifying the mappings are commonly used: local-as-view (LAV) and global-as-view (GAV). The LAV formalism relates each element in sources to a query over the ontology, while the GAV formalism associates each element of the ontology to a query over the sources. Since an ontology is regarded as a standard conceptualization of a domain and tends to be stable, the LAV formalism prevails. Similar to the LAV formalism in the traditional data integration systems, a typical mapping formula associates an element in sources to a conjunctive formula over concepts and relationships in an ontology as shown in the Formula 1.1. A major difference, however, exists. An ontology is usually an object-oriented description, while a data source often is relational, therefore value-oriented. A reconciliation between objects and tuples of values sometimes is needed in mapping specification. The DWQ [CGL+01a] data integration system originally used the notion of *adornment* to express the correspondence between a tuple of values and the object it represents.

The DWQ system is aimed at integrating data from different sources into a materialized data warehouse through an ontology. Mappings from data sources and the warehouse to the ontology enable the automatic creation of the mediator for loading data into the warehouse. The ontology is described in an enriched Entity-Relationship model. Both sources and the warehouse are linked to the ontology by adorned mapping formulas. In particular, an adorned mapping formula is an expression of the form

$$T(\vec{x}) \rightarrow q(\vec{x}, \vec{y}) \mid \alpha_1, ..., \alpha_n \tag{2.2}$$

where the head $T(\vec{x})$ defines a table in a relational source in terms of a name T, and its arity, i.e., the number of columns, the body $q(\vec{x}, \vec{y})$ describes the content of the table in terms of an ontology,

and $\alpha_1, ..., \alpha_n$ constitutes the adornment in which each α_i is an annotation on variables appearing in $\vec{x}$. The body $q(\vec{x}, \vec{y})$ is a union of conjunctions of atoms. Each atom is a concept, a relationship, or an attribute appearing in the ontology. In the adornment, there are two types of annotations. For each $X \in \vec{x}$, an annotation $X::V$ specifies the domain of a column of the table T. For each tuple of variables $\vec{z} \subseteq \vec{x}$ that is used for identifying in T an object $Y \in \vec{y}$ mentioned in $q(\vec{x}, \vec{y})$, an annotation is of the form $Identify([\vec{z}], Y)$.

Example 2.1.2. Suppose a university ontology contains two concept, Student and Professor, and a relationship hasAdvisor between the two concepts. Both Student and Professor as subclasses of Person inherit three attributes: ssn for social security number, dob for date of birth, and name. Suppose a data source $\mathcal{S}_1$ contains the information about the students and their supervisors, in terms of a relational table supervisor(sname, sdob, pname, pdob), where sname is for student's name, sdob for student's birth date, pname for supervisor's name, and pdob for supervisor's birth date. A data source $\mathcal{S}_2$ contains the information about the professors and their supervised students, in terms of a relational table supervision(sSsn, pSsn), where sSsn and pSsn are for student's and professor's social security numbers, respectively. We assume that in the source $\mathcal{S}_1$ persons (both students and professors) are identified by their name and date of birth, while in $\mathcal{S}_2$ persons are identified by their social security number. Using adorned formulas, we can specify the mappings from the sources to the ontology as follows:

supervisor($sname$, $sdob$, $pname$, $pdob$) $\rightarrow$ Student(X_1), Professor(X_2),
hasAdvisor(X_1, X_2),
name(X_1, $sname$), dob(X_1, $sdob$),
name(X_2, $pname$), dob(X_2, $pdob$)
| $sname, pname$ :: NameString,
$sdob, pdob$ :: Date,
$Identify([sname, sdob], X_1)$,
$Identify([pname, pdob], X_2)$.

supervision($sSsn$, $pSsn$) $\rightarrow$ Student(X_1), Professor(X_2),
hasAdvisor(X_1, X_2),
ssn(X_1, $sSsn$), ssn(X_2, $pSsn$),
| $sSsn, pSsn$ :: SSNString,
$Identify([sSsn], X_1)$,
$Identify([pSsn], X_2)$.

■

Aside from the mappings linking data sources and the warehouse to the ontology, there are other reconciliation correspondences for resolving heterogeneity in creating the mediator in the DWQ system. Three types of correspondences are used, namely, Conversion, Matching, and Merging correspondences. As the number of the data sources increases and ontologies become more complex, it is desirable to have automatic tools for generating the mapping formulas and the reconciliation correspondences. Such a tool not only is useful for the DWQ integration system, but also is applicable to a wide variety of scenarios involving mapping databases to CMs. One of the major contributions of this dissertation is the development of such an automatic tool for discovering the semantic mapping, similar to the mappings in Example 2.1.2, from a database schema to a CM, e.g., an ontology.

It is natural to ask whether we can apply the RIC-based techniques to derive the semantic mappings from database schemas to CMs, when a CM is viewed as a relational database consisting

of tables for concepts, attributes, and relationships. Our investigations show that the RIC-based techniques do not produce the desired results in many cases. This is partly due to the fact that the semantic mapping asks for "semantically similar" associations in terms of modeling a subject matter. Another reason is that the RIC-based techniques sometimes miss desired associations in a single schema. Concrete examples will be shown in following chapters. The bulk of our work in this dissertation focuses on the problem of discovering such a pair of "semantically similar" associations for different mapping tasks, based partly on the principles of conceptual modeling.

In addition to tools for schema mapping, a considerable body of work exists for discovering mappings between ontologies [KS03a, PS05]. Current ontology mapping tools, however, only focus on deriving simple correspondences between concepts or between properties, despite ontologies having more complex structures and capturing more real-world knowledge than database schemas. Example tools are PROMPT [NM03], FCA-merge [SM01], IF-Map [KS03b], and GLUE [DMDH02]. Because of the simple form of their mapping results, we classify the tools as finding correspondences and discuss some of them in the next section. Ontology mappings are used in ontology translation [DMQ03], ontology merging [NM03], and ontology integration [CGL01b].

2.2 Finding Correspondences: Schema and Ontology Matching

To improve the chances of getting more accurate and reliable mappings, many mapping discovery tools as well as the mapping discovery algorithms developed in this dissertation take a set of element correspondences as an extra input in addition to the schemas being mapped. The set of element correspondences can be specified manually by users. More desirable, if correspondences could be generated automatically by some tools. In this section, we take a glimpse of the body of work related to finding correspondences, which is referred to as *schema matching* or *ontology matching*. What we are concerned about is the underlying meaning of a correspondence generated by a schema matching tool: What does the *algorithm* tell us about an element in one schema corresponding to an element in another schema.

Although *schema matching* is also referred to as the problem of identifying semantic relation-

ships between schemas [RB01, PS05] by some researchers, the major difference from the *schema mapping* we termed in the previous section is that the results of schema matching often are a set of correspondences each of which links a single element in one schema to a single element in another schema. Sometimes, however, a correspondence may involve a function relating a single element to several elements [DLD$^+$04], for example, salary = hourly_wage $\times$ working_hours, or vice versa. In particular, a matcher takes two schemas as input and produces a matching between elements of the two schemas that correspond "semantically" to each other. Matchers are classified into schema- and instance-level, element- and structure-level, language- and constraint-level, and individual and combination methods. It is worth noting that almost all methods proposed so far in the literature [DNH04, NDH05] are semi-automatic and need the intervention of the user. Just given two database schemas without other explicit connections between them, few claim that they can fully automatically discover correct matchings without user's examination on final results.

Specifically, Cupid [MBR01] discovers correspondences between schema elements based on their names, data types, constraints, and schema structures. Using directed graph as the unifying internal representation of a spectrum of schemas, Cupid strives to identify the *similarity* between elements. This is accomplished in two steps: Linguistic-based matching follows structure-based matching. The assumption in Cupid about "semantic correspondence" is that the names of the elements convey linguistic similarity and the structures of the representation help to propagate similarity along graph edges. Matching is computed, essentially, by using heuristics. The technique of Similarity Flooding [MGMR02] converts schemas into directed labeled graphs as well and uses fixpoint computation to determine the matches between corresponding nodes of the graphs. For ontologies with rich semantics, Anchor-PROMPT [NM01b] takes as input a set of anchors – pairs of related terms in two ontologies, producing pairs of other related terms. Anchor-PROMPT also assumes the structure conveys similarity. The COMA [DR02] does not invent any new matching algorithm. Instead it develops a framework to combine multiple matchers in a flexible way. Multiple matching techniques can be plugged in COMA system to produce composite matching results for input schemas. Hence the "semantic" interpretation of schema elements depends on underlying matchers. LSD and GLUE [DDH01, DMDH02] are systems that employ machine learning tech-

niques to find similar elements. They use multiple learners exploiting various types of knowledge about data types, names, instances, structures, and previous matchings. Realizing that evidence carried by schemas themselves are not sufficient to derive more accurate matching results, Xu et al. in [XE03b, XE03a] utilize domain ontologies in matching discovery and Madhavan et al. in [MBDH05] explore a corpus of schemas for evaluating the similarity of a pair of elements.

For light-weight schemas like web service descriptions and web query interfaces, statistical techniques play a critical role in discovering matchings between web services [DHM+04] and matchings between web interfaces [HC06].

Most of the schema matching tools do not formally define the problem they are dealing with. They do not state clearly what it means for two elements to be most similar. An exception is the study presented in [Doa02], where the author makes the assumptions underlying the matching algorithms explicit. In short, the common underlying assumptions behind many schema matching tools can be summarized as follows. Given two models $\mathcal{S}$ and $\mathcal{T}$, that for an element $e_1 \in \mathcal{S}$, there is an element $e_2 \in \mathcal{T}$ corresponding to e_1 means that the real world object represented by e_2 has the highest similarity to the real world object represented by e_1, compared to all other real world objects represented by all other elements in $\mathcal{T}$. The matching results of such a schema matching tool is in fact an approximation of the true objects in the real world described by the elements of domain models and an estimation of the true similarity between objects in real world domains.

2.3 Model Management

Model management [BHP00, BR00, Ber03] is a framework for providing a generic management facility for formal descriptions of complex application artifacts such as relational database schemas, XML schemas/DTDs, conceptual models, web interface definitions, web service descriptions, and workflow diagrams. These formal descriptions are referred to as *models*. A novel feature in model management is that both models and mappings between models are treated as abstractions that can be manipulated by model-at-a-time and mapping-at-a-time operators. Here lies the claim that an implementation of these abstractions and operators could offer an order-of-magnitude improvement

in productivity for metadata management. Formally, a *model* is defined to be a set of objects, each of which has properties, has-a relationships, and associations. A *model* is assumed to be identified by its root object and includes exactly the set of objects reachable from the root by paths of has-a relationships. Given two models M_1 and M_2, a *morphism* over M_1 and M_2 is a binary relation over the objects of the two models. That is, it is a set of pairs $\langle o_1, o_2 \rangle$ where o_1 and o_2 are in M_1 and M_2, respectively. A *mapping* between models M_1 and M_2 is a model, map_{12}, and two morphisms, one between map_{12} and M_1 and another between map_{12} and M_2.

The literature suggests six major model management operators, match creating a mapping between two models, compose combining two successive mappings into one, merge merging two models into a third model using a mapping between the two models, extract returning a portion of a model that participates in a mapping, diff returning a portion of a model that does not participating in a mapping, and confluence merging two mappings. The first prototype implementing some of the operators, called Rondo, is presented in [MRB03]. The implementation, however, treats mappings as syntactic links without rigorous semantics. Therefore, for the match operator in Rondo, existing schema matching tools provide a solution. Since merging models such as schema integration lies at the heart of many metadata applications, the generic merge operator is studied in [PB03]. The main contributions are the precise definition of the semantics of the merge operator, the classification of the conflicts that arise in merging models, and the resolution strategies for conflicts that must be resolved in merge. Furthermore, the merge operator along with other four operators, extract, diff, compose, and confluence, are studied in [MBHR05] when mappings are specified as executable statements in some formal languages.

It should be noted that the match operator in model management can produce both syntactic links and executable statements depending on the particular application. Our work in this dissertation amounts to the match operator for generating executable mappings. As we have reviewed in Section 2.1, most previous solutions focus on producing executable mappings between database schemas by solely exploring evidence in schemas. We now study a match operator implementation taking as the input database schemas and CMs, and we further propose a semantic approach for implementing the match operator for database schemas.

2.4 Database Reverse Engineering (DBRE)

Database reverse engineering (DBRE) [Hai98] is another relevant area of research. Even though there are similarities between this area and the problem we are tackling, there are also important differences. First, DBRE is aimed at extracting a CM from a logical database schema, while our work attempts to recover the semantics of a schema using a given CM, though not necessarily exactly the semantic data model from which the logical schema may have been built. Second, different from graph construction algorithms in DBRE, the semantics recovery algorithm in our work heavily employs the graph-theoretic algorithms for discovering "reasonable" connections in a conceptual model graph. Third, the results of our algorithm are logical formulas representing semantics of a database schema, while DBRE produces just a pictorial representation of a conceptual model, without showing how it links to the database schema.

Since many databases and their operational environments evolve constantly, maintaining these databases has long been recognized as a painful and complex activity. An important aspect of database maintenance is the recovery of a CM that represents the meaning of the logical schema. Database reverse engineering (DBRE) is defined as the process of recovering such a CM by examining an existing database system to identify the database's contents and their interrelationships.

Approaches to translating a relational database schema into a conceptual schema have appeared since the beginning of the 1980s [Cas83, DA83, NA87, MM90]. Gradually, four main sources have been explored for finding evidence to construct a conceptual schema from a logical database: the structures and integrity constraints of the database schema, the application programs that access the database, the data instances stored in the database, and the users and designers. Specifically, the algorithm in [FG92] uses only schema structures and constraints focusing on the subtype/supertype relationships which are created at an early stage in the algorithm. Andersson's work in [And94] reengineers legacy systems, where the only information provided by the DBMS is table names, field names, indices, and possibly view definitions. Information about functional dependencies, keys, and inclusion dependencies are deduced by looking into data manipulation statements that can be extracted from the application code. The approach in [CBS94] analyzes not only the database schema,

but also data instances which contain detailed information about the application domain. Additionally, specific kinds of SQL queries are analyzed in [PKBT94] for helping to build an Extended Entity-Relationship schema including ISA relationships and aggregates; the techniques of program understanding which emerged in the Software Engineering field are employed in [HEH+98] to improve understanding the domain semantics of database schemas.

Similar to our problem of discovering semantics of database schemas in terms of given CMs, DBRE is also difficult to automate and needs human intervention. Since the sources used for deriving evidence to construct a new conceptual model do not contain sufficient semantic information, the conceptual models created by many DBRE methods are often closely tied to the existing database schemas and hence may become, in worst case, just graphical representations of the actual logical and physical implementations of the databases. To deal with complex DBRE problems more effectively, researchers have adopted a traditional Computer Science technique: divide and conquer. In [SdJPA02], the complex problem of dealing with a large database system is divided into smaller problems: Relations are directly grouped into elements of a high-level abstract schema, and intermediate conceptual models are constructed for each group. Finally, the intermediate conceptual models are consolidated into a single conceptual model with missing elements. Moreover, for some specific types of constructs in logical schemas as well as in conceptual models, specialized reverse engineering methods are developed. For example, star schemas are reengineered in [KS97], while n-ary relationships and aggregate relationships are extracted in [Sou96] and [Sou98], respectively.

Most of the DBRE methods we have reviewed are informal. In particular, they rely on various heuristics to generate elements in a conceptual model from available sources and do not formally specify the quality of the results. Two possible criteria are “correctness” and “faithfulness”. Correctness concerns whether an element in the conceptual model extracted from a construct of a logical schema or database represents the intended meaning of the construct. Faithfulness measures the degree to which all information in the logical schema and database have been represented in the conceptual model and nothing else. The work of [Joh94] proposes a method for translating relational schemas into conceptual schemas. The method decides the correct object types in the conceptual schemas by interacting with users. Furthermore, it argues that the method proposed is faithful by

showing that the conceptual schemas produced are able to represent the same information as the original logical schemas. One of the contributions of this dissertation is that we provide formal results for a subclass of problems, although the algorithm is necessarily heuristic. The formal results show that the algorithm is "correct" for the database schemas that were derived from conceptual models by the standard database design methodology.

2.5 Query Processing Over Mappings

Our work is also related to the study of query processing over mappings. Not only is the ultimate goal of mapping discovery answering queries against the information systems containing mappings, but also rewriting queries over mappings is an important step in discovering schema mappings using semantics of schemas. In this section we review some of the achievements that have been made in query processing over mappings.

As defined in Section 2.1, a data integration system involves a global schema $\mathcal{G}$, a source schema $\mathcal{S}$, and the mapping $\mathcal{M}$ between $\mathcal{G}$ and $\mathcal{S}$. The goal of a data integration system is to provide a unified view of the data stored in the sources so that queries about the sources can be posed against the global schema and the queries will be answered by the system which shields users from knowing the location and detailed description of each source. Query processing in a data integration system involves different manners in terms of LAV or GAV modeling approaches. In LAV, the problem of processing a query is traditionally called *view-based query processing* which is classified into *view-based query rewriting* and *view-based query answering*.

Query rewriting is aimed at reformulating the original query in terms of a query to the sources in a query language $\mathcal{L}_\mathcal{Q}$, in a way that is independent of the current source databases. Sometimes, no equivalent rewriting exists in the fixed target query language $\mathcal{L}_\mathcal{Q}$. In this case, people are interested in computing a so-called *maximally containment rewriting*. A rewritten query Q' w.r.t. a query language $\mathcal{L}_\mathcal{Q}$ is maximally contained in the original query means that there are no other rewritten queries w.r.t. $\mathcal{L}_\mathcal{Q}$ that contain Q'. A comprehensive survey [Hal01] discusses the large body of work on algorithms for query rewritings. For conjunctive view definitions and conjunctive queries,

important query rewriting algorithms are the Bucket algorithm [LRO96], the Minicon algorithm [PH01], and the Inverse rule algorithm [Qia96, DGL00]. For Datalog queries, the rewriting is also approached by the Inverse rule algorithm [DGL00]. For conjunctive queries with arithmetic comparisons, an algorithm is developed in [ALM02]. In ontology-based information integration systems, queries are posed against the ontology which acts as the global schema. Query rewriting incorporates the reasoning about the ontology [GR04]. Finally, when the mapping between XML data are specified as source-to-target tuple generating dependencies, a variant of the inverse rule algorithm is developed in [YP04] for rewriting queries posed to the target into queries over the source.

When developing algorithms that produce rewritten queries, one can ask two questions [Hal00]: (i) whether the algorithms is sound and complete: given a query Q and a set of views $\mathcal{V}$, is there an algorithm that will find a rewriting of Q using $\mathcal{V}$ when one exists; (ii) what is the complexity of that problem. For the class of queries and views expressed as conjunctive queries, the study in [LMSS95] shows that when the query does not contain comparison predicates and has n subgoals, then there exists an equivalent conjunctive rewriting of Q using $\mathcal{V}$ only if there is a rewriting with at most n subgoals. An immediate corollary is that the problem of finding an equivalent rewriting of a query using a set of views is in NP, because it suffices to guess a rewriting and check its correctness. Furthermore, the work in [Hal00] also points out that the problem of finding a contained rewriting is NP-complete.

As we can see, the maximally-contained query rewriting needs to check for query containment. Here, we refer to the literature on query containment. NP-completeness for conjunctive queries is established in [CM77]. Π_2^p- completeness of containment of conjunctive queries with inequalities is proved in [Klu88, vdM92]. The case of queries with the union and difference operators is studied in [SY80]. Results of the decidability and undecidability of various classes of Datalog queries with inequalities are presented in [CV92, vdM92]. The problem of answering queries using views is closely related to the problem to query rewriting. The complexities of answering queries using views under various view definition languages and query languages are given in [AD98].

Query processing through the GAV mappings can be a simple unfolding strategy if there are

no integrity constraints in the global schema and the views are exact. However, when the global schema allows integrity constraints, and the views are sound, then query processing in GAV systems becomes more complex. The simple unfolding algorithm does not retrieve all certain answers in the presence of integrity constraints of the global schema [CCGL02]. Nevertheless, for foreign key constraints, all certain answers can be computed [CCGL02] by expanding queries with the constraints and using the partial evaluation of logic programming, in the case that the language for expressing both the user query and the queries in the mapping is the one of union of conjunctive queries.

We will study a semantic approach to discovering schema mapping expressions assuming semantics of schemas are available. The semantics of each schema is specified in terms of a set of semantic mapping formulas each of which relates a predicate/expression representing a basic organizational unit in the schema to a conjunctive formula over a CM. The approach first discovers a graphical connection in the CM graph and then translates the discovered connection into an expression over the schema. Consider the discovered connection as a query over the CM graph. The translation problem becomes a query rewriting problem, where special provisions are needed to reconcile the object identifiers in CM world with tuples in databases.

2.6 Conceptual Modeling and Data Semantics

We finally discuss conceptual modeling and data semantics. Conceptual modeling is concerned with the construction of computer-based symbol structures which model some part of the real world directly and naturally [Myl98]. Conceptual modeling originated from several areas in Computer Science [BMS84]. In Artificial Intelligence, conceptual modeling is concerned with *knowledge representation* which is the problem of capturing human knowledge so that it can be used by a software system. In Database, conceptual modeling produces *semantic data models* which are used to directly and naturally model an application before proceeding to a logical and physical database design. In Programming Languages, conceptual modeling is concerned with different forms of *abstraction* which allow implementation-independent specifications of data, functions, and controls.

For data management, semantic data models offer more semantic terms and abstraction mechanisms for modeling an application than logical data models. A logical data model usually provides abstract mathematical symbol structures for hiding the implementation details which are the concerns of a physical data model. The Entity-Relationship model [Che75] assumes that an application consists of *entities* and *relationships*. Entities and relationships have *attributes*. This ontological assumption is intended to make the Entity-Relationship model more expressive, i.e., capable of capturing more world semantics than the relational model [Cod70]. Specifically, a set of entities having the same characteristics (the same set of attribute names) are modeled as an *entity set* in an ER model. A *relationship set* models a set of similar relationships. A *key* is a minimal set of attributes whose values uniquely identify an entity in an entity set. Sometimes, a key may consist of some attributes of other entities. To specify the occurrence of an entity in a certain type of relationship, one can use the *cardinality constraints*. In order to capture more semantics, the Extended Entity-Relationship (EER) model [MS92] introduces generalization/specialization relationships between entity sets and allows relationships to participate in other relationships.

Figure 2.3 is a typical EER diagram which models a university domain. It has four entity sets: Student, GradStudent, Professor, and Course and two relationship sets: supervisedBy and teaches. GradStudent is a subclass of Student. In terms of the participations of entities in the relationships, a student participates in the supervisedBy relationship at most once, meaning a student can have at most one supervisor; there is not limitation for a professor to participate in the same relationship set. On the other hand, a professor must teach at least one course (the minimum cardinality for participating in the teaches relationship is 1), while a course can be taught by one professor.

In 1966, Ross Quillian [Qui68] proposed in his PhD thesis *semantic networks*, a form of directed, labeled graphs, as a convenient device for modeling the structure of human memory. Nodes of his semantic network represented concepts (more precisely, word sense). Nodes were related through links representing semantic relationships, such as ISA, has, and other relationships. The semantic networks were proposed for serving as a general inferential representation for knowledge. The inference techniques was based on the notion of a spreading activation intersection search – given

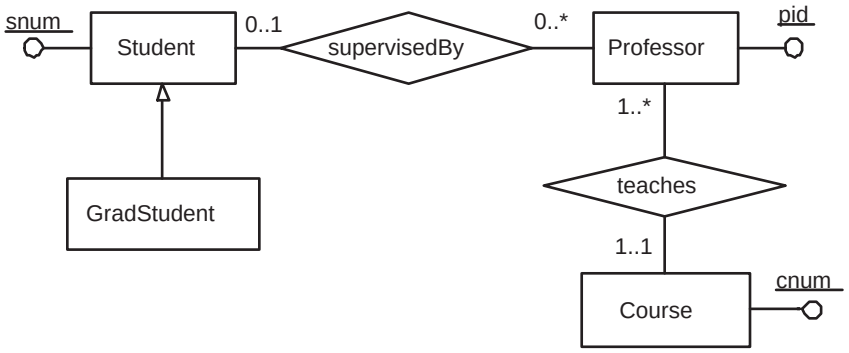

Figure 2.3: An Extended Entity-Relationship Diagram

two words, possible relations between them might be inferred by an unguided, breadth-first search of the area surrounding the words.

Inferring potential relationships between words by using graphical connections was one of the important features of semantic networks. Our work on discovering semantics of database schemas in terms of given CMs was initially inspired by Quillian's work in semantic networks. In particular, we have a set of concepts (singled out by the simple correspondences linking elements in a database schema to attributes of concepts) in a CM. What we want is some "reasonable" connection among these concepts such that the connection possibly matches the semantics of the schema. A significant difference is that we attempt to find a connection in a CM to *match* an association (among a set of elements) in a logical model. This requires not only discovering all potential connections but also finding those that are "correct".

The Unified Modeling Language (UML) was proposed by a consortium in the late 1990s and soon became a standard modeling tool for Object-Oriented software design. The UML class diagram models static aspects of an application. For example, a database designer can use a class diagram modeling the data. A UML class diagram offers the following abstraction mechanisms: class, association, generalization, and composition. Specifically, a composition specifies a *partOf* association. A "whole" class is made up of component classes. A strong form of aggregation is composite, where a component in a composite can be part of only one whole. Aggregations and composites are represented as lines joining the whole and the component with open and filled di-

amond, respectively, on the whole side. Figure 2.4 shows a UML class diagram for modeling a university domain. Notice that the memberOf association is also a composition association; cardinality constraints are specified at the opposite directions compared to the EER diagram in Figure 2.3.

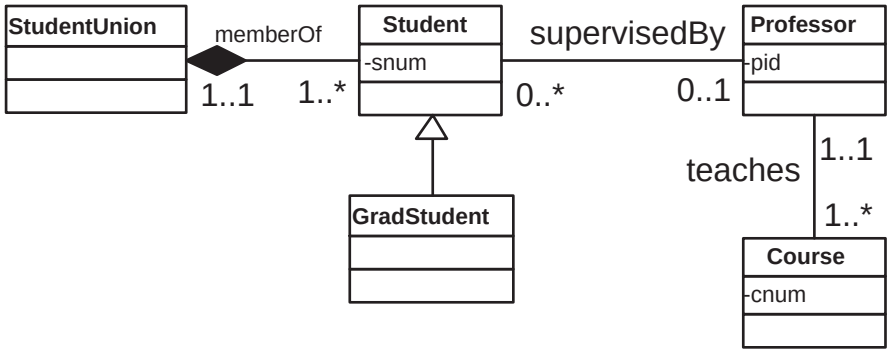

Figure 2.4: A UML Class Diagram

Since most semantic networks lack formal semantics as pointed by Woods [Woo75], there have been attempts to integrate ingredients from formal logic to semantic models. An early example of this trend was terminological language such as KL-One [BS85]. Later, the family of Description Logics (DL) grew out of this. Generally speaking, Description Logics are formalisms that represent the knowledge of an application domain (the "world") by first defining the relevant concepts of the domain (its terminology), and then using these concepts to specify properties of objects and individuals occurring in the domain (the world description). Elementary descriptions are atomic concepts and atomic roles/binary relationships. Complex descriptions can be built from them inductively using concept constructors. Different Description Languages are distinguished by the constructors they provide. Concept descriptions in $\mathcal{AL}[\mathcal{U}][\mathcal{E}][\mathcal{N}][\mathcal{C}][\mathcal{I}]$ are formed according to the syntax rules which may allow union ($\mathcal{U}$), full existential quantification ($\mathcal{E}$), number restriction ($\mathcal{N}$), negation of arbitrary concept ($\mathcal{C}$), and inverse role ($\mathcal{I}$). The semantics of Description Logics are specified by the model theory, and algorithms are sought for deciding judgments such as concept subsumption and consistency [B$^+$02].

DLs are also the underlying formalism of the Semantic Web. The idea behind applying DLs to

the semantic web is related to the need of representing and reasoning on ontologies. The OWL web ontology language [MvH04] is intended to define and instantiate web ontologies.

The fundamental concern for all semantic/conceptual models is the abstraction mechanisms. By definition, abstraction involves suppression of (irrelevant) details. Abstraction mechanisms organize the data stored in a database, providing semantics and guiding the use of the data. In the following, we summarize some common abstraction mechanisms used in the semantic models surveyed above. A important ingredient of our mapping discovery algorithms is to find "similar" abstraction mechanisms in different models.

Classification, sometimes called *instanceOf*, classifies instances under one or more generic classes. Instances of a class share common properties. For example, all instances classified under Person have an address and an age. Some information models allow classification to be recursive, i.e., classes may themselves be instances of other classes. Telos [MBJK90] and RDF [KC03] are two such examples.

Generalization, referred to as *ISA*, organizes all classes in terms of a partial order relation determined by their generality/specificity. Inheritance is a functional inference rule of generalization mechanism.

Aggregation, also called *partOf*, views objects as aggregates of their components or parts. A strong form of aggregation states that a component can be a part of only one aggregate.

As we discussed at the beginning of this dissertation, CMs were mainly used during database design-time and subsequently converted into logical schemas in the data model manipulated by the underlying DBMS. Since the main focus of a DBMS is performance, a logical schema uses concise mathematical structures to represent various abstraction mechanisms in CMs. This had worked fine for a closed and relatively stable operational environment. In this dissertation, we are interested in the problem of *recovering* mappings from logical schemas to CMs and using these mappings for facilitating data integration and exchange in an open, dynamic, and distributed environment. In the next chapter, we begin with the detailed description of the problem.

Chapter 3

Problem Description

We define the problem of mapping discovery in this chapter. We first describe database schemas and CMs. Two types of schemas are introduced, relational and XML. Both are commonly used for describing data in databases. Next, we identify and describe three specific mapping discovery problems, namely, mapping from relational schemas to CMs, mapping from XML schemas to CMs, and mapping between relational database schemas. These specific problems will be considered in depth in the chapters that follow.

3.1 Schemas and CMs

A database schema is a description of data in terms of a data model which contains a set of abstract and high-level constructs for describing an application. Primary examples of data models include the relational model, the XML data model, and the object-oriented model. For the purpose of this dissertation, we will focus on relational schemas, XML schemas, and CMs in terms of the generic CML.

3.1.1 Relational Model and Relational Schema

The basic data description construct of the relational model is relation, which can be thought of as a set of rows. The schema for a relation or a *table* specifies the name of the relation, the name of each column (or attribute or field), and the type of each column. For example, Figure 3.1 shows a relation with the following schema:

Employee(empno:$Integer$, name:$String$, dept: $String$, proj: $String$)

empno	name	dept	proj
288566345	Jones	Research & Development	P000234
288566346	Smith	Research & Development	P000234
288566348	Smith	Marketing	P000234
288566359	Alice	Finance	P000234
288566360	Peter	Product	P000387

Figure 3.1: A Relational Table

This table contains a set of tuples each of which describes an employee using values under the specified attributes. Furthermore, we can make the description of the collection of employees more precise by specifying *integrity constraints*, which are conditions that the tuples in the table must satisfy. For example, we could require that every employee have a unique empno value. A subset of columns that uniquely identifies a tuple is called *key* in a table.

A relational schema consists of a set of relational tables. Formally, we use the notation $T(\underline{K}, Y)$ to represent a relational table T with columns KY, and key K. For individual columns in Y, we refer to them as $Y[1], Y[2], \ldots$, and use XY as concatenation of columns. In the rest of the dissertation, our notational convention for relational schemas is that single column names are either indexed or appear in lower-case. Given a table such as T above, we use the notation key(T), nonkey(T) and columns(T) to refer to K, Y and KY respectively. Other important constraint in the relational model which plays a critical role in our mapping discovery process is the foreign key constraint. Specifically, a A foreign key (abbreviated as *f.k.* henceforth) in T is a set of columns F that *references* the key of table T', and imposes a constraint that the projection of T on F is a subset of the projection of T' on key(T').

3.1.2 XML Data Model and XML Schema

An XML document is typically modeled as a node-labeled graph. For our purpose, we assume that each XML document is described by an XML schema consisting of a set of element and attribute type definitions. Specifically, we assume the following countably infinite disjoint sets: **Ele** of element names, **Att** of attribute names, and **Dom** of simple type names including the built-in XML schema datatypes. Attribute names are preceded by a "@" to distinguish them from element names. Given finite sets $E \subset$**Ele** and $A \subset$**Att**, an XML schema $\mathcal{S} = (E, A, \tau, \rho, \kappa)$ specifies the type of each element ℓ in E, the attributes that ℓ has, and the datatype of each attribute in A. Specifically, an element type τ is defined by the grammar $\tau ::= \epsilon|\mathsf{Sequence}[\ell_1 : \tau_1, ...\ell_n : \tau_n]|\mathsf{Choice}[\ell_1 : \tau_1, .., \ell_n : \tau_n]$ for $\ell_1, .., \ell_n \in E$, where ϵ is for the empty type, and Sequence and Choice are complex types. Each element associates an occurrence constraint with two values: *minOccurs* indicating the minimum occurrence and *maxOccurs* indicating the maximum occurrence. (We mark with * multiply occurring elements.) The set of attributes of an element $\ell \in E$ is defined by the function $\rho : E \rightarrow 2^A$; and the function $\kappa : A \rightarrow$**Dom** specifies the datatypes of attributes in A. Each datatype name associates with a set of values in a domain Dom. In this dissertation, we do not consider the *simple type elements* (corresponding to DTD's **PCDATA**), assuming instead that they have been represented using attributes. All attributes are *single-valued.* Furthermore, a special element $\underline{r} \in E$ is the root of each XML schema, and we assume that for any two element $\ell_i, \ell_j \in E$, $\rho(\ell_i) \cap \rho(\ell_j) = \emptyset$.

For example, an XML schema describing articles and authors has the following specification:
$E =\{article, author, contactauthor, name\}$,
$A =\{@title, @id, @authorid, @fn, @ln\}$,
$\tau(article)$ = Sequence$[(author)* :\tau(author), contactauthor{:}\epsilon]$,
$\tau(author)$ = Sequence$[name{:}\epsilon]$,
$\rho(article) = (@title)$, $\rho(author) = (@id)$, $\rho(contactauthor) = (@authorid)$,
$\rho(name) = (@fn, @ln)$, $\kappa(@title)$ = String, $\kappa(@authorid)$ = Integer, $\kappa(@id)$= Integer, $\kappa(@fn)$= String, $\kappa(@ln)$= String, and the element $article$ is the root. Note that for the *article* element,

contactauthor only occurs once, while *author* may occur many times. For the *author* element, *name* occurs once. This XML schema can be described in the XML Schema Language [FW04] as shown in Figure 3.2. The XML Schema Language is an expressive language that can also express key and keyref constraints.

```
<xsd:element name=''article'' type=''articleType''/>
  <xsd:complexType name=''articleType''>
    <xsd:sequence>
      <xsd:element name=''author''>
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name=''name'' minOccurs=''1''
                maxOccurs=''1''>
              <xsd:complexType>
                <xsd:attribute name=''fn''
                 type=''xsd:string'' use=''required''/>
                <xsd:attribute name=''ln''
                 type=''xsd:string'' use=''optional''/>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
          <xsd:attribute name=''id'' type=''xsd:integer''
           use=''required'' />
        </xsd:complexType>
      </xsd:element>
      <xsd:element name=''contactauthor'' minOccurs=''1''
           maxOccurs=''1''>
        <xsd:complexType>
          <xsd:attribute name=''authorid''
           type=''xsd:integer'' use=''required'' />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name=''title'' type=''xsd:string''
     use=''required'' />
  </xsd:complexType>
</xsd:element>
```

Figure 3.2: An XML Schema Description

Unlike relational databases where data are stored in relations comprising tuples of values, data in

XML documents are organized in graph (tree) structures. An XML document $\mathcal{X} = (N, <, \underline{r}, \lambda, \eta)$ over (E, A) consists of a set of nodes N, a child relation $<$ between nodes, a root node $\underline{r}$, and two functions such as:

- a labeling function λ:$N \rightarrow E \cup A$ such that if $\lambda(v) = \ell \in E$, we say that v is in the element type ℓ; if $\lambda(v) = @a \in A$, we say that v is an attribute $@a$;
- a partial function η:$N \rightarrow Dom$ for every node v with $\lambda(v) = @a \in A$, assigning values in domain Dom that supplies values to simple type names in **Dom**.

An XML document $\mathcal{X} = (N, <, \underline{r}, \lambda, \eta)$ conforms to a schema $\mathcal{S} = (E, A, \tau, \rho, \kappa)$, denoted by $\mathcal{X} \models \mathcal{S}$, if:

1. for every node v in $\mathcal{X}$ with children $v_1, .., v_m$ such that $\lambda(v_i) \in E$ for $i = 1, ..., m$, if $\lambda(v) = \ell$, then $\lambda(v_1)$,..., $\lambda(v_m)$ satisfies $\tau(\ell)$ and the occurrence constraints.
2. for ever node v in $\mathcal{X}$ with children $u_1, ..., u_n$ such that $\lambda(u_i) = @a_i \in A$ for $i = 1, ..., n$, if $\lambda(v) = \ell$, then $\lambda(u_i) = @a_i \in \rho(\ell)$, and $\eta(u_i)$ is a value having datatype $\kappa(@a_i)$.

An XML schema can be viewed as a directed node-labeled graph called *schema graph* consisting of the following edges: parent-child edges $e = \ell \rightarrow \ell_i$ for elements $\ell, \ell_i \in E$ such that if $\tau(\ell)$= Sequence$[...\ell_i : \tau_i...]$ or Choice$[...\ell_i : \tau_i...]$; and attribute edges $e = \ell \Rightarrow \alpha$ for element $\ell \in E$ and attribute $\alpha \in A$ such that $\alpha \in \rho(\ell)$. For a parent-child edge $e = \ell \rightarrow \ell_i$, if the $maxOccurs$ constraint of ℓ_i is 1, we show the edge to be functional, drawn as $\ell \Rightarrow \ell_i$. Since attributes are single-valued, we always draw an attribute edge as $\ell \Rightarrow \alpha$. The schema graph corresponding to the XML schema in Figure 3.2 is shown in Figure 3.3.

Elements and attributes as nodes in a schema graph are located by path expressions. For our purposes, we use a simple path expression $Q = \epsilon|\ell.Q$ and introduce the notion of *element tree*.

An *element tree* represents an XML structure whose semantics we are seeking for. A semantic mapping from an XML schema to a CM consists of a set of mapping formulas each of which is from an element tree to a conjunctive formulas in the CM. An *element tree* can be constructed through

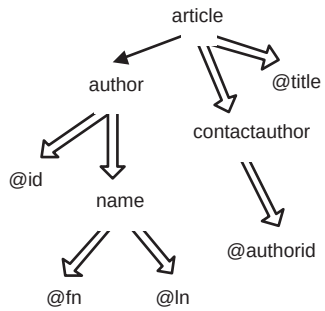

Figure 3.3: An XML Schema Graph

doing a depth first search (DFS), starting from the element node for which we are constructing an element tree. The DFS process first creates an *element graph* and goes as follows. Begin with an empty element graph and create a new node in the element graph for each unmarked original node during the traversal of the original schema graph. Mark each node in the schema graph as "visited" when it is reached at the first time and unmarked when all of its descendent's have been traversed. Regular edges are created in the element graph whenever there is a traversal from a DFS parent node to its unmarked children in the original schema graph. If an already marked node is being traversed, then a "back" edge (using dashed line) is created in the element graph from the DFS parent to this marked child. For example, Figure 3.4 (a) shows a schema graph with a cycle. Figure 3.4 (b), (c), and (d) are the element graphs created by the DFS process starting at the elements controls, employee, and manager, respectively.

Next, we convert the element graphs into element trees by ignoring or unfolding the back edges depending on our needs. To unfold a back edge from a node ℓ_i to a node ℓ_j, we connect ℓ_j and all the contents descending ℓ_j until ℓ_i to ℓ_i and remove the back edge. The occurrence constraint of the new created edge from ℓ_i to ℓ_j is the same as that of the back edge. Figure 3.5 (c) and (d) are the element trees converted from the element graphs in Figure 3.4 (c) and (d), respectively, by unfolding the back edges, while Figure 3.5 (b) is the element tree converted from the element graph in Figure 3.4 (b) by ignoring the back edge. For the sake of simplicity, we specify each element tree as rooted at the element from which the tree is constructed, ignoring the path from the root to the element in the original schema graph.

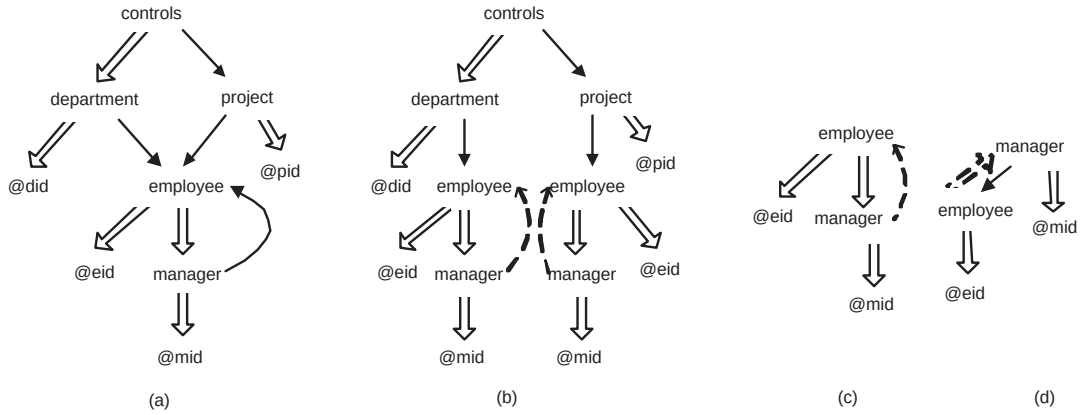

Figure 3.4: Schema Graph and Element Graphs

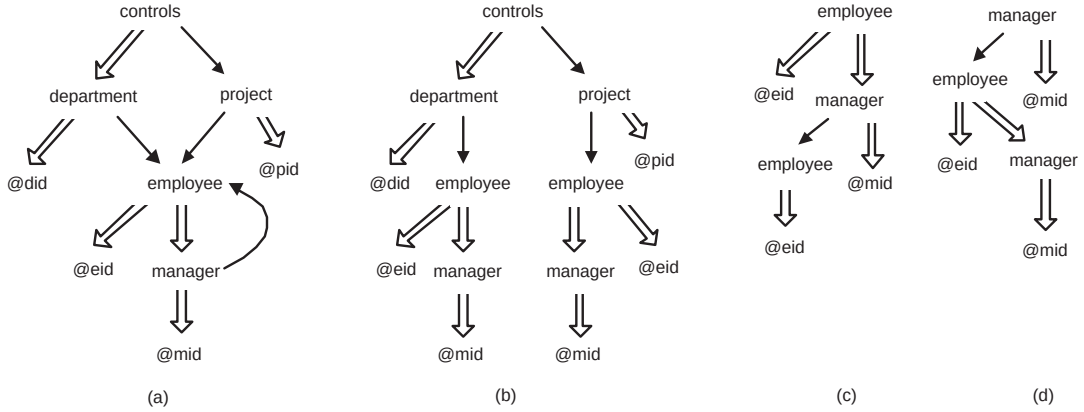

Figure 3.5: Schema Graph and Element Trees

3.1.3 CMs and the CM Graph

In this dissertation, we do not restrict ourselves to any particular language for describing CMs. Instead, we use a generic conceptual modeling language (CML), which contains *common* aspects of most semantic data models, UML, ontology languages such as OWL, and description logics. Specifically, the language allows the representation of *classes/concepts* (unary predicates over individuals), *object properties/relationships* (binary predicates relating individuals), and *datatype properties/attributes* (binary predicates relating individuals with values such as integers and strings); attributes are single valued in this dissertation. Concepts are organized in the familiar ISA hierar-

chy, and subclasses of a superclass can be either disjoint or overlapping. Relationships, and their inverses (which are always present), are subject to constraints such as specification of domain and range, plus cardinality constraints, which here allow 1 as lower bounds (called *total* relationships), and 1 as upper bounds (called *functional* relationships).

We shall represent a given CM using a labeled directed graph, called an *CM graph*. We construct the CM graph from a CM as follows: We create a concept node labeled with C for each concept C, and an edge labeled with p from the concept node C_1 to the concept node C_2 for each object property p with domain C_1 and range C_2; for each such p, there is also an edge in the opposite direction for its inverse, referred to as p^-. For each attribute f of concept C, we create a separate attribute node denoted as $N_{f,C}$, whose label is f, and add an edge labeled f from node C to $N_{f,C}$. For each ISA edge from a subconcept C_1 to a superconcept C_2, we create an edge labeled with ISA from concept node C_1 to concept node C_2 with cardinality 1..1 on the C_2 side (a C_1 must be a C_2), and 0..1 on the C_1 side. For the sake of succinctness, we sometimes use UML notations, as in Figure 3.6, to represent the CM graph. Note that in such a diagram, instead of drawing separate attribute nodes, we place the attributes inside the rectangle concept nodes; and relationships and their inverses are represented by a single undirected edge. The presence of such an undirected edge, labeled p, between concepts C and D will be written in text as C ---p--- D. It will be important for our approach to distinguish *functional edges* – ones with upper bound cardinality of 1, and their composition: *functional paths*. If the relationship p is functional from C to D, we write C ---p->-- D. For expressive CMLs such as OWL, we may also connect C to D by p if we find an existential restriction stating that each instance of C is related to *some* instance or *only* instances of D by p.

3.2 Mapping Discovery Problems

We now identify and describe the specific mapping discovery problems we consider in this dissertation. We first describe the problem of mapping a relational schema to a CM. Second, we define the problem of discovering a mapping from an XML schema to a CM. We specify a mapping for-

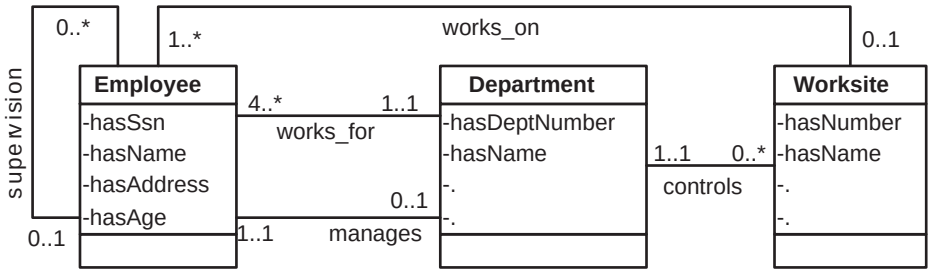

Figure 3.6: A CM Graph in UML Notation

malism and discuss why the problem is different from the problem for relational schema. Finally, we describe the problem of discovering mappings between relational schemas and contrast it with the other two problems.

3.2.1 The Problem of Mapping Relational Schemas to CMs

In this setting, we want to discover a semantic mapping from a relational schema to a CM, given a set of correspondences. A *correspondence* $T.c \leftrightsquigarrow D.f$ relates column c of a relational table T to an attribute f of a concept D of a CM. Since our algorithms deal with CM graphs, formally a correspondence L will be a mathematical relation $L(T, c, D, f, N_{f,D})$, where the first two arguments determine unique values for the last three. This means that a table column corresponds to a single attribute of a concept (complex correspondences associating with multiple attributes can be treated as union of a set of correspondences each of which associates with a single attribute).

We use formulas in the form $T(X) \rightarrow \Phi(X, Y)$, as described in Section 1.1, to represent the *semantic mapping* from a relational schema to a CM, where T is a table with columns X (which become arguments to its predicate), and Φ is a conjunctive formula over predicates representing the CM.

Problem 1 (R-to-O problem). *Given a relational schema $\mathcal{R} = \langle \mathcal{T}, \Delta \rangle$ with a set of relational table $\mathcal{T}$=$\{T_1,...,T_n\}$ and a set of integrity constraints Δ, a CM $\mathcal{O}$, and a set of correspondences L from columns of tables in $\mathcal{T}$ to attributes of concepts in $\mathcal{O}$. For a table $T_i(X)$, find an association $\delta_{\mathcal{T}}$ in*

the CM $\mathcal{O}$ such that $T_i(X)$ and $\delta_{\mathcal{T}}$ are "semantically similar" in terms of modeling a subject matter.

The input of problem 1 (R-to-O problem) is a relational schema, a CM, and a set of correspondences from columns of tables in the schema to attributes of concepts in the CM. A relational table stores attribute values organized into tuples, while a CM specifies concepts, attributes of concepts, and relationships between concepts. A close relationship between relational schemas and CMs is induced by the standard database design methodology which consist of a set of principles that convert a conceptual description into a logical schema. Our solution is to find an association/subgraph in the CM graph by examining the structures and integrity constraints in the schema. The goal is that after converting the association/subgraph into a table by the standard design methodology, the table is indistinguishable from the original table under consideration in the schema in terms of structures and constraints.

The output of problem 1 is a set of mapping formulas. These candidate formulas need to be examined by the user. Of course, it is desirable that the candidate list is "complete" meaning that it contains all expected formulas and that the list is as short as possible so that user would spend the minimum amount of effort to examine the list. To evaluate the performance of our solution, we manually create mapping formulas for each tested dataset in our experiments. These manually create mapping formulas serve as the "gold standard" for testing the achievement of semantic similarity. We compare the results generated by the solution with these correct formulas. We check how many correct ones are contained in the candidate list. Furthermore, if an expected formula is not generated by the solution, we measure how much effort has to be put into correcting an "incorrect" one into a correct one. We choose the above method because it is sufficient to quantify the effectiveness and usefulness of the solution.

3.2.2 The Problem of Mapping XML schemas to CMs

In this setting, we attempt to discover a semantic mapping from an XML schema to a CM, given a set of simple correspondences. A *correspondence* $P.@c \rightsquigarrow D.f$ relate the attribute "$@c$" of an

element ℓ reached by the simple path P to the datatype property f of the class D in the CM. A simple path P is always relative to the root of the tree. For example, we can specify the following correspondences for the element tree in Figure 3.5 (c):

$employee.@eid1 \leftrightsquigarrow$ Employee.hasId,

$employee.manager.@mid \leftrightsquigarrow$ Employee.hasId.

$employee.manager.employee.@eid2 \leftrightsquigarrow$ Employee.hasId

where Employee is a concept in a CM and hasId is an attribute of the concept Employee. As in the relational case, formally, a correspondence L will be a mathematical relation $L(P, @c, D, f, N_{f,D})$, where the first two arguments determine unique values for the last three.

We now turn to the mapping language relating a formula representing an element tree with a conjunctive formula in a CM. On the XML side, the basic components are *attribute formulas* [AL05], which are specified by the syntax $\alpha ::= \ell | \ell(@a_1 = x_1, .., @a_n = x_n)$, where $\ell \in E$, $@a_1, .., @a_n \in A$; E and A are element names and attribute names, respectively, while variables $x_1, .., x_n$ are the free variables of α. Tree-pattern formulas over an XML schema $\mathcal{S} = (E, A, \tau, \rho, \kappa)$ are defined by $\psi ::= \alpha | \alpha[\varphi_1, .., \varphi_n]$, where α ranges over attribute formulas over (E, A). The free variables of a tree formula ψ are the free variables in all the attribute formulas that occur in it. For example, $employee(@eid1 = x_1)[manager(@mid = x_2)[employee(@eid2 = x_3)]]$ is the tree formula representing the element tree in Figure 3.5 (c).

A *mapping formula* between an element tree and a CM then has the form $\Psi(X) \rightarrow \Phi(X, Y)$, where $\Psi(X)$ is a tree formula in the XML schema and $\Phi(X, Y)$ is a conjunctive formula in the CM. For example, given a CM containing a concept Employee, with an attribute hasId, and a functional property hasManager (whose inverse is manages, which is not functional), the following mapping formula ascribes a semantics of the element tree in Figure 3.5 (c):

$employee(@eid1 = x_1)$[
$manager$ ($@mid = x_2$)[
$employee$ ($@eid2 = x_3$)]] $\rightarrow$ Employee(Y_1),hasId(Y_1, x_1), Employee(Y_2),
hasId(Y_2, x_2), hasManager(Y_1, Y_2),
Employee(Y_3), hasId(Y_3, x_3),manages(Y_2, Y_3).

Since we maintain the unique name assumption for attributes, we can drop the variable names x_is, and just use attribute names in formulas. The variables Y_js are implicitly existentially quantified and refer to individuals in the CM.

Problem 2 (X-to-O problem). *Given an XML schema $\mathcal{S} = (E, A, \tau, \rho, \kappa)$, a CM $\mathcal{O}$, and a set of correspondences L from attributes of elements in $\mathcal{S}$ to attributes of concepts in $\mathcal{O}$. For an element tree T, find an association δ_T in the CM $\mathcal{O}$ such that T and δ_T are "semantically similar" in terms of modeling a subject matter.*

The input of the X-to-O problem is an XML schema, a CM, and a set of correspondences from attributes of elements in the schema to datatype properties of concepts in the CM. An XML document stores attribute values organized into a graph, while a CM specifies concepts, attributes of concepts, and relationships between concepts. As for the relational case, our solution for discovering the semantic mapping from an XML schema to a CM also exploits the principles that convert a CM into a "good" XML schema. Focusing on semantics discovery, we assume the input XML schema has been transformed into element tree(s).

We now discuss the differences between the X-to-O problem and the R-to-O problem. Much research has focused on converting and storing XML data into relational databases [STH$^+$99]. It is natural to ask whether we could utilize the mapping algorithm we have developed for the R-to-O problem by first converting XML DTDs/schemas into relational tables or by applying the mapping algorithm directly to XML schemas. Unfortunately, this approach does not work. First of all, the algorithms that generate a relational schema from an XML DTD/schema use backlinks and system

generated *id*s in order to record the nested structure, and these confuse the algorithms for the R-to-O problem, which rely heavily on key and foreign key information for semantics of real world objects. Secondly, an XML schema is a rooted graph (tree) structure. Much semantics are encoded in the parent-child links as well as the occurrence constraints imposed on the links. The principles for designing an XML schema from a CM is different from that for designing a relational schema from a CM; therefore, when we seek for "semantically similar" associations in XML schemas and CMs, the algorithm will be different from that for relational schemas and CMs. Thirdly, the outputs are different and need different treatments. One relates a tree formula to a conjunctive formula, while other relates a table as an atomic formula to a conjunctive formula. Finally, and most generally, the X-to-O problem is different from the R-to-O problem because XML schemas and relational schemas are heterogeneous domain models subscribing to different modeling languages. Although they may describe the same subject matter, they use different modeling constructs for different purposes. A particular algorithm for one model often does not produce desired results for another model because the algorithm has been designed for exploiting specific modeling constructs. We believe that an effective tool for discovering semantics of different models has to employ different algorithms geared to particular modeling languages.

The output of problem 2 is a set of candidate mapping formulas. We use manually created "correct" mapping formulas for each tested dataset to evaluate the performance of the solution. The performance is measured in terms of recall, precision, and labor savings.

3.2.3 The Problem of Mapping Database Schemas to Database Schemas

In this setting, we want to discover semantic mapping from a source relational schema $\mathcal{S}$ to a target relational schema $\mathcal{T}$, given a set of correspondences from columns of $\mathcal{S}$ to columns of $\mathcal{T}$. A *correspondence* $R.c \leftrightsquigarrow T.f$ will relate the column c of the table R in $\mathcal{S}$ to the column f of the table T in $\mathcal{T}$. For the set L of correspondences between the schema $\mathcal{S}$ and the schema $\mathcal{T}$, we use $L(\mathcal{S})$ and $L(\mathcal{T})$ to denote the sets of columns linked by L in $\mathcal{S}$ and $\mathcal{T}$, respectively.

The goal of schema mapping is to find an association among columns in $L(\mathcal{S})$ and an association

among columns in $L(\mathcal{T})$ such that the pair of associations are semantically similar in terms of modeling a subject matter. For relational schemas, we use join algebraic expressions for associations among columns as shown in the following example:

S_1: gradStudent(sno, sname, pid)$\bowtie$professor(pid, pname, area).

Given a pair of algebraic expressions, one could derive mapping formulas in the form s-t tgd and multiple mapping expressions could be derived. For example, suppose we have another algebraic expression which is a single table

T_1: student(sno, name, advisor, area),

and we want to derive mapping formula from the pair $\langle S_1, T_1 \rangle$ given the correspondences

v_1: gradStudent.sno$\leftrightsquigarrow$student.sno,

v_2: gradStudent.sname$\leftrightsquigarrow$student.name,

v_3: professor.pname$\leftrightsquigarrow$student.advisor,

v_4: professor.area$\leftrightsquigarrow$student.area.

Here are three possibilities:

M_1: $\forall sno, sname, pid.$(gradStudent(sno, $sname$, pid) $\rightarrow$ student(sno, $name$, _, _)).

M_2: $\forall sno, sname, pid, pname, area.$(gradStudent($sno$, $sname$, pid)$\wedge$
professor(pid, $pname$, $area$) $\rightarrow$ student(sno, $sname$, $pname$, $area$)).

M_3: $\forall pid, pname, area.$(professor($pid$, $pname$, $area$) $\rightarrow$ student(_, _, $pname$, $area$)).

The mapping expression M_1 covers the correspondences v_1 and v_2, M_2 covers v_1, v_2, v_3, and v_4, and M_3 covers v_3 and v_4. Since the derivation depends on usage of the mapping and other constraints, we will leave it open and specify a mapping as a 3-tuple $\langle E_1, E_2, L_M \rangle$, where E_1 and E_2 are algebraic expressions in the source and target, respectively, and L_M is a set of correspondences that are covered by the pair of expressions. Specifically, a correspondence linking a source column c to a target column f is covered by a pair $\langle E_1, E_2 \rangle$ of expressions, if c appears in E_1 and f appear in E_2.

Problem 3 (R-to-R problem). *Given a relational schema $\mathcal{S}$ associated with a CM $\mathcal{G}_S$ through a semantic mapping Σ_S and a relational schema $\mathcal{T}$ associated with a CM $\mathcal{G}_T$ through a semantic mapping Σ_T. Let L be the set of correspondences linking a set $L(\mathcal{S})$ of columns in $\mathcal{S}$ to a set $L(\mathcal{T})$ of columns in $\mathcal{T}$. For an algebraic expression E_1 connecting columns in $L(\mathcal{S})$, find an algebraic expression E_2 connecting columns in $L(\mathcal{T})$ such that E_1 and E_2 are "semantically similar" in terms of modeling a subject matter.*

By our solution to the problem 1 (R-to-O problem), we assume that the semantic mappings relate each table in the schemas to a semantic tree in the respective CM graphs, and with each column we can associate a (unique) concept node in a graph through associations from columns to attribute nodes. Consequently, the set $L(\mathcal{S})$ of columns gives rise to a set $\mathcal{C}_S$ of concept nodes in the graph $\mathcal{G}_S$. Likewise, the set $L(\mathcal{T})$ gives rise to a set $\mathcal{C}_T$ of concept nodes in the graph $\mathcal{G}_T$. We call the semantic trees which contain nodes in $\mathcal{C}_S$ and $\mathcal{C}_T$ *pre-selected s-trees*. Our mapping discovery process consists of two major steps: (1) finding a conceptual subgraph (hereafter CSG) D_S connecting nodes in $\mathcal{C}_S$ and a CSG D_T connecting nodes in $\mathcal{C}_T$ such that D_S and D_T are "semantically similar" (we have stripped off the attribute nodes temporarily); (2) restoring the attribute nodes used to identify the concept nodes and translating D_S into an algebraic expression E_1 and D_T into an algebraic expression E_2. The pair $\langle E_1, E_2 \rangle$ is returned as a mapping candidate if it covers the set of L or a subset of L.

Compared to the R-to-O and X-to-O problems, the R-to-R problem uses the results of the R-to-O problem for improving traditional schema mapping solutions. To evaluate the performance of our solution to the R-to-R problem, we compare the results of our solution to that of the traditional RIC-based techniques. The comparison is made against the manually created mapping formulas as the "correct" ones. The performance is measured in terms of recall and precision.

3.3 Summary

In this chapter, we first presented database schemas and CMs. A database schema is a description of data in terms of a data model. In this dissertation, we focus on two commonly used database schemas, the relational schema describing data in terms of the relational model and the XML schema describing data in terms of the XML data model. We create a CM graph from a CM described in a generic CML language. Next, we defined three specific mapping discovery problems, the R-to-O problem, the X-to-O problem, and the R-to-R problem. We specified the inputs, outputs, principles for the solutions, and evaluation methods for these problems. In the next three chapters, we develop solutions for these specific problems and evaluate the solutions using comprehensive sets of test data drawn from a variety of application domains.

Chapter 4

Discovering Semantic Mappings from Relational Schemas to CMs

We now begin to develop a solution for discovering semantic mappings from relational schemas to CMs. We first describe the problem in Section 4.1. Next, we present an intuitive progression of ideas underlying our approach in Section 4.2. In the two sections that follow, we provide a mapping inference algorithm in Section 4.3 and report on the prototype implementation of these ideas and experimental results in Section 4.4. In Section 4.5, we discuss the issues of generating GAV mapping formulas from the LAV formulas produced by our solution. Finally, we discuss the limitations of our solution and future work in Section 4.6 and summarize the chapter in Section 4.7.

4.1 The Problem

We use formulas in the form $T(X) \rightarrow \Phi(X, Y)$ to represent semantic mappings from relational schemas to CMs. A semantic mapping formula relates $T(X)$, a single predicate representing a table in a relational schema, to $\Phi(X, Y)$, a conjunctive formula over the predicates representing the concepts and relationships in a CM.

Recall that manual creation of mapping formulas is difficult, time-consuming and error-prone,

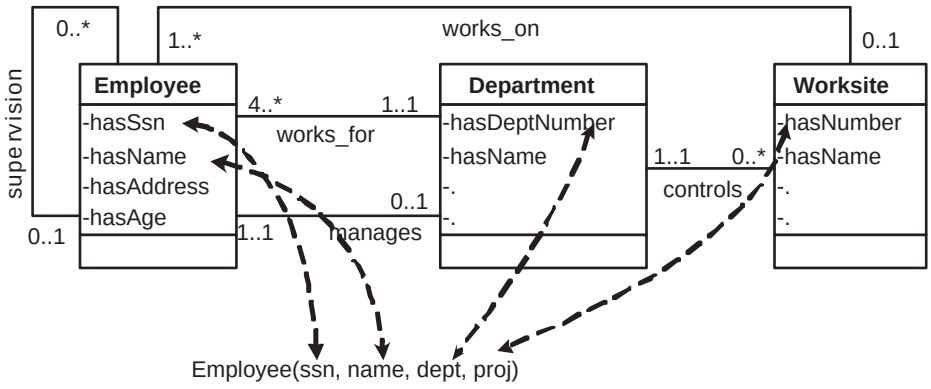

Figure 4.1: Relational table, CM, and Correspondences.

we propose a tool that assists users in specifying mapping formulas. In order to improve the effectiveness of our tool, we expect the tool user to provide *simple correspondences* between atomic elements used in the database schema (e.g., column names of tables) and those in the CM (e.g., datatype property/attribute names of concepts). Given the set of correspondences, the tool is expected to reason about the database schema and the CM, attempting to find "semantically similar" pairs of associations. At last, it generates a list of candidate formulas for each table in the relational database. Ideally, one of the formulas is the correct one — capturing user intention underlying given correspondences. The following example illustrates the input/output behavior of the tool proposed.

Example 4.1.1. Figure 4.1 contains the enterprise CM we have demonstrated in Chapter 1. Suppose we wish to discover semantics of a relational table Employee(ssn,name, dept, proj) with key ssn in terms of the enterprise CM. Suppose that by looking at column names of the table and the CM graph, the user draws the simple correspondences shown as dashed arrows in Figure 4.1. This indicates, for example, that the ssn column corresponds to the hasSsn property of the Employee concept. Using prefixes $\mathcal{T}$ and $\mathcal{O}$ to distinguish tables in the relational schema and concepts in the CM, we represent the correspondences as follows:

$\mathcal{T}$:Employee.ssn$\leftrightsquigarrow\mathcal{O}$:Employee.hasSsn

$\mathcal{T}$:Employee.name$\leftrightsquigarrow\mathcal{O}$:Employee.hasName

$\mathcal{T}$:Employee.dept$\leftrightsquigarrow\mathcal{O}$:Department.hasDeptNumber

$\mathcal{T}$:Employee.proj$\leftrightsquigarrow\mathcal{O}$:Worksite.hasNumber

Given the above inputs, the tool is expected to produce a list of plausible mapping formulas, which would hopefully include the following formula, expressing a possible semantics for the table:

$$\begin{aligned}\mathcal{T}\text{:Employee}(ssn, name, dept, proj) \quad \rightarrow \quad & \mathcal{O}\text{:Employee}(x_1),\ \mathcal{O}\text{:hasSsn}(x_1,ssn),\\ & \mathcal{O}\text{:hasName}(x_1,name),\ \mathcal{O}\text{:Department}(x_2),\\ & \mathcal{O}\text{:works\_for}(x_1,x_2),\ \mathcal{O}\text{:hasDeptNumber}(x_2,dept),\\ & \mathcal{O}\text{:Worksite}(x_3),\ \mathcal{O}\text{:works\_on}(x_1,x_3),\\ & \mathcal{O}\text{:hasNumber}(x_3,proj).\end{aligned}$$

■

An intuitive (but somewhat naive) solution, inspired by early work of Quillian [Qui68], is based on finding the *shortest* connections between concepts. Technically, this involves (i) finding the minimum spanning tree(s) (actually Steiner trees[1]) connecting the "corresponded concepts" — those that have datatype properties corresponding to table columns, and then (ii) encoding the tree(s) into formulas. However, in some cases the spanning/Steiner tree may not provide the desired semantics for a table because of known relational schema design rules.

Example 4.1.2. Consider the relational table Project(name, supervisor), where the column name is the key and corresponds to the attribute $\mathcal{O}$:Worksite.hasName, and column supervisor corresponds to the attribute $\mathcal{O}$:Employee.hasSsn in Figure 4.1. The minimum spanning tree consisting of Worksite, Employee, and the edge works_on probably does not match the semantics of table Project because there are multiple Employees working on a Worksite according to the CM cardinality, yet the table allows only one to be recorded, since supervisor is functionally dependent on

[1] A Steiner tree [HRW92] for a set M of nodes in graph G is a minimum spanning tree of M that may contain nodes of G which are not in M.

name, the key. Therefore we must seek a functional connection from Worksite to Employee, and the connection will be the manager of the department controlling the worksite. ■

Since our approach is directly inspired by the Clio project [MHH00, PVM+02], which developed a successful tool that infers mappings from one set of relational/XML schemas to another, given just a set of correspondences between their respective attributes, it is natural to ask whether Clio's current solution is sufficient for the problem. To apply Clio's technique, we could view the present problem as extending Clio to the case where the source schema is a relational database while the target is a CM. The next example, however, shows that Clio's solution does not produce the desired results.

Example 4.1.3. We can view the CM in Figure 4.1 as a relational schema made of unary tables for the concepts and binary tables for the attributes and relationships. Specifically, there are three unary tables Employee(x_1), Department(x_2), and Worksite(x_3) for the three concepts. Moreover, for the attributes of the concepts and the relationships between concepts, there are a number of binary tables including

hasSsn(x_1, ssn), hasName($x_1, name$), works_for(x_1, x_2),
hasDeptNumber($x_2, dept$), hasNumber($x_3, proj$), and works_on(x_1, x_3)...

The obvious foreign key constraints are from binary to unary tables, e.g.,

works_for.$x_1 \subseteq$ Employee.x_1,
works_for.$x_2 \subseteq$ Department.x_2...

Then one could in fact try to apply directly the Clio's current algorithm to the problem.

Recall that Clio's current algorithm works by taking each table and using a chase-like algorithm to repeatedly extend it with columns that appear as foreign keys referencing other tables. Such "logical relations" in the source and target are then connected by queries. In this particular case, this would lead to logical relations such as

$$\mathsf{Employee}(x_1) \bowtie \mathsf{works\_for}(x_1, x_2) \bowtie \mathsf{Department}(x_2),$$
$$\mathsf{Employee}(x_1) \bowtie \mathsf{works\_on}(x_1, x_2) \bowtie \mathsf{Department}(x_2),$$
$$\mathsf{Employee}(x_1) \bowtie \mathsf{hasSsn}(x_1),$$
$$\mathsf{Employee}(x_1) \bowtie \mathsf{hasName}(x_1),$$
$$\mathsf{hasDeptNumber}(x_2) \bowtie \mathsf{Department}(x_2),$$
$$\mathsf{hasNumber}(x_3) \bowtie \mathsf{Worksite}(x_3).$$

The desired mapping formula in Example 4.1.1 would not be produced because none joins $\mathsf{hasSsn}(x_1, ssn)$ and $\mathsf{hasDeptNumber}(x_2, dept)$ through some intermediary, which is part of the desired formula.

The fact that ssn is a key for the table $\mathcal{S}$:Employee, leads us to prefer a many-to-one relationship, such as works_for, over some many-to-many relationship which could have been part of the CM (e.g., $\mathcal{O}$:previouslyWorkedFor); Clio does not differentiate the two. So the work to be presented here analyzes the key structure of the tables and the semantics of relationships (cardinality, IsA) to eliminate/downgrade *unreasonable* options that arise in mappings to CMs. Furthermore, our principles of mapping inference exploit largely the knowledge of database design in seeking "semantically similar" associations.

■

In this dissertation, we use ideas of standard relational schema design from ER diagrams in order to craft heuristics that systematically uncover the connections between the constructs of relational schemas and those of CMs. We propose a tool to generate "reasonable" trees connecting the set of concepts in a CM which have attributes participating in the given correspondences. In contrast to the graph theoretic results which show that there may be too many minimum spanning/Steiner trees among a set of concept nodes in a CM (for example, there are already 5 minimum spanning trees connecting Employee, Department, and Worksite in the very simple graph in Figure 4.1), we expect the tool to generate only a small number of "reasonable" trees. These expectations are born out by our experimental results, in Section 4.4.

4.2 Principles of Mapping Discovery

Given a table T, and correspondences L to a CM provided by a person or a tool, let the set $\mathcal{C}_T$ consist of those concept nodes which have at least one attribute corresponding to some column of T, i.e., D such that there is at least one tuple $L(\_, \_, D, \_, \_)$. Our task is to find meaningful associations between concepts in $\mathcal{C}_T$. Attributes can then be connected to the result using the correspondence relation: for any node D, one can imagine having edges f to M, for every entry $L(\_, \_, D, f, M)$. The primary principle of our mapping discovery algorithm is to look for *smallest* "reasonable" trees connecting nodes in $\mathcal{C}_T$. We will call such a tree a *semantic tree*.

As mentioned before, the naive solution of finding minimum spanning trees or Steiner trees does not give good results, because it must also be "reasonable". We aim to describe more precisely this notion of "reasonableness".

Consider the case when $T(\underline{c}, b)$ is a table with key c, corresponding to an attribute f on concept C, and b is a foreign key corresponding to an attribute e on concept B. Then for each value of c (and hence instance of C), T associates at most one value of b (instance of B). Hence the semantic mapping for T should be some formula that acts as a function from its first to its second argument. The semantic trees for such formulas look like functional edges in the CM, and hence are more reasonable. For example, given table Dep(dept,ssn, ...), and correspondences

$\mathcal{T}$:Dep.dept $\leftrightsquigarrow \mathcal{O}$:Department.hasDeptNumber,

$\mathcal{T}$:Dep.ssn $\leftrightsquigarrow \mathcal{O}$:Employee.hasSsn,

from the table columns to attributes of the CM in Figure 4.1, the proper semantic tree uses manages$^-$ (i.e., hasManager) rather than works_for$^-$ (i.e., hasWorkers).

Conversely, for table $T'(\underline{c}, \underline{b})$, where c and b are as above, an edge that is functional from C to B, or from B to C, is likely not to reflect a proper semantics since it would mean that the key chosen for T' is actually a super-key – an unlikely error. (In our example, consider a table $T(\underline{ssn, dept})$, where both columns are foreign keys.)

To deal with such problems, our algorithm works in two stages: first connects the concepts

corresponding to key columns into a *skeleton tree*, then connects the rest of the corresponded nodes to the skeleton by functional edges (whenever possible).

We must however also deal with the assumption that the relational schema and the CM were developed independently, which implies that not all parts of the CM are reflected in the database schema. This complicates things, since in building the semantic tree we may need to go through additional nodes, which end up not corresponding to columns of the relational table. For example, consider again the table Project(name, supervisor) and its correspondences mentioned in Example 4.1.2. Because of the key structure of this table, based on the above arguments we will prefer the functional *path* controls$^-$.manages$^-$ (i.e., controlledBy followed by hasManager), passing through node Department, over the shorter path consisting of edge works_on, which is not functional. Similar situations arise when the CM contains detailed *aggregation* hierarchies (e.g., city part-of township part-of county part-of state), which are abstracted in the database (e.g., a table with columns for city and state only).

We have chosen to flesh out the above principles in a systematic manner by considering the behavior of our proposed algorithm on relational schemas designed from Entity Relationship diagrams — a technique widely covered in undergraduate database courses [RG02]. (We refer to this er2rel *schema design*.) One benefit of this approach is that it allows us to prove that our algorithm, though heuristic in general, is in some sense "correct" for a certain class of schemas. Of course, in practice such schemas may be "denormalized" in order to improve efficiency, and, as we mentioned, only parts of the CM may be realized in the database. Our algorithm uses the general principles enunciated above even in such cases, with relatively good results in practice. Also note that the assumption that a given relational schema was designed from some ER conceptual model does not mean that given CM is this ER model, or is even expressed in the ER notation. In fact, our heuristics have to cope with the fact that it is missing essential information, such as keys for weak entities.

To reduce the complexity of the algorithms, which essentially enumerate all trees, and to reduce the size of the answer set, we modify a CM graph by collapsing multiple edges between nodes E and F, labeled $p_1, p_2, \ldots$ say, into at most three edges, each labeled by a string of the form $'p_{j_1}; p_{j_2}; \ldots'$: one of the edges has the names of all functions from E to F; the other all functions from F to E;

and the remaining labels on the third edge. (Edges with empty labels are dropped.) Note that there is no way that our algorithm can distinguish between semantics of the labels on one kind of edge, so the tool offers all of them. It is up to the user to choose between alternative labels, though the system may offer suggestions, based on additional information such as heuristics concerning the identifiers labeling tables and columns, and their relationship to property names.

4.3 Semantic Mapping Discovery Algorithms

As mentioned, our algorithm is based in part on the relational database schema design methodology from ER models. We introduce the details of the algorithm iteratively, by incrementally adding features of an ER model that appear as part of the CM. We assume that the reader is familiar with basics of ER modeling and database design [RG02], though we summarize the ideas.

4.3.1 ER_0: An Initial Subset of ER notions

We start with a subset, ER_0, of ER that supports entity sets E (called just "entity" here), with attributes (referred to by $\mathsf{attribs}(E)$), and binary relationship sets. In order to facilitate the statement of correspondences and theorems, we assume in this section that attributes in the CM have globally unique names. (Our implemented tool does not make this assumption.) An entity is represented as a concept/class in our CM. A binary relationship set corresponds to two properties in our CM, one for each direction. Such a relationship is called *many-many* if neither it nor its inverse is functional. A *strong entity* S has some attributes that act as identifier. We shall refer to these using $\mathsf{unique}(S)$ when describing the rules of schema design. A *weak entity* W has instead $\mathsf{localUnique}(W)$ attributes, plus a functional total binary relationship p (denoted as $\mathsf{idRel}(W)$) to an identifying owner entity (denoted as $\mathsf{idOwn}(W)$).

Example 4.3.1. An ER_0 diagram is shown in Figure 4.2, which has a weak entity Dependent and three strong entities: Employee, Department, and Project. The owner entity of Dependent is Employee and the identifying relationship is dependents_of. Using the notation we introduced, this means that

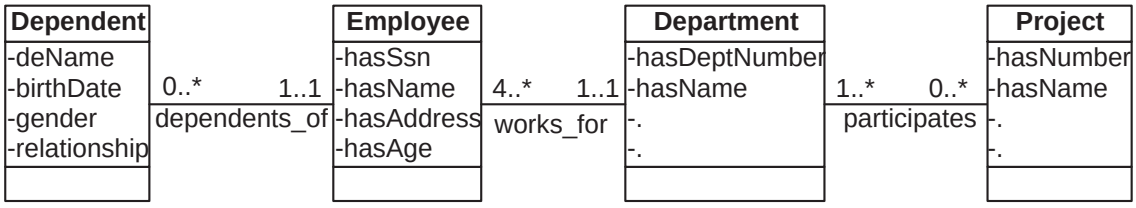

Figure 4.2: An ER_0 Example.

localUnique($Dependent$) =$deName$, idRel($Dependent$)= $dependents\_of$, idOwn($Dependent$)= $Employee$. For the owner entity Employee, unique($Employee$)= $hasSsn$.

■

Note that information about multi-attribute keys cannot be represented formally in even highly expressive CM languages such as OWL. So functions like unique are only used while describing the er2rel mapping, and are not assumed to be available during semantic inference. The er2rel design methodology (we follow mostly [MM90, RG02]) is defined by two components. To begin with, Table 4.1 specifies a mapping $\tau(O)$ returning a relational table scheme for every CM component O, where O is either a concept/entity or a binary relationship. (For each relationship exactly one of the directions will be stored in a table.)

In addition to the schema (columns, key, f.k.'s), Table 4.1 also associates with a relational table $T(V)$ a number of additional notions:

- an *anchor*, which is the central object in the CM from which T is derived, and which is useful in explaining our algorithm (it will be the root of the semantic tree);
- a formula for the semantic mapping for the table, expressed as a formula with head $T(V)$ (this is what our algorithm should be recovering); in the body of the formula, the function hasAttribs(x, Y) returns conjuncts $attr_j(x, Y[j])$ for the individual columns $Y[1], Y[2], \ldots$ in Y, where $attr_j$ is the attribute name corresponded by column $Y[j]$.
- the formula for a predicate identify$_C(x, Y)$, showing how object x in (strong or weak) entity

ER Model object O	Relational Table τ(O)	
Strong Entity S	*columns:*	X
	primary key:	K
Let X=attribs(S)	*f.k.'s:*	none
Let K=unique(S)	*anchor:*	S
	semantics:	$T(X) \rightarrow S(y)$,hasAttribs(y, X).
	identifier:	identify$_S(y, K) \rightarrow S(y)$,hasAttribs$(y, K)$.
Weak Entity W	*columns:*	ZX
let	*primary key:*	UX
$E =$ idOwn(W)	*f.k.'s:*	X
$P =$ idrel(W)	*anchor:*	W
Z=attribs(W)	*semantics:*	$T(X, U, V) \rightarrow W(y)$, hasAttribs$(y, Z)$, $E(w)$,$P(y, w)$, identify$_E(w, X)$.
$X =$ key($\tau(E)$)		
$U =$localUnique(W)	*identifier:*	identify$_W(y, UX) \rightarrow W(y)$,$E(w)$, $P(y, w)$, hasAttribs(y, U), identify$_E(w, X)$.
$V = Z - U$		
Functional	*columns:*	X_1X_2
Relationship F	*primary key:*	X_1
$\boxed{E_1}$ - - F - > - $\boxed{E_2}$	*f.k.'s:*	X_i references $\tau(E_i)$,
let $X_i =$ key($\tau(E_i)$)	*anchor:*	E_1
for $i = 1, 2$	*semantics:*	$T(X_1, X_2) \rightarrow E_1(y_1)$,identify$_{E_1}(y_1, X_1)$, $F(y_1, y_2)$, $E_2(y_2)$, identify$_{E_2}(y2, X_2)$.
Many-many	*columns:*	X_1X_2
Relationship M	*primary key:*	X_1X_2
$\boxed{E_1}$ - - M - - $\boxed{E_2}$	*f.k.'s:*	X_i references $\tau(E_i)$,
let $X_i =$ key($\tau(E_i)$)	*semantics:*	$T(X_1, X_2) \rightarrow E_1(y_1)$,identify$_{E_1}(y_1, X_1)$, $M(y_1, y_2)$,$E_2(y_2)$, identify$_{E_2}(y2, X_2)$.
for $i = 1, 2$		

Table 4.1: er2rel Design Mapping.

C can be identified by values in Y[2].

Note that τ is defined recursively, and will only terminate if there are no "cycles" in the CM (see [MM90] for definition of cycles in ER).

Example 4.3.2. When τ is applied to concept Employee in Figure 4.2, we get the table

$\mathcal{T}$:Employee(hasSsn, hasName, hasAddress, hasAge),

with the anchor Employee, and the semantics expressed by the mapping:

[2]This is needed in addition to hasAttribs, because weak entities have identifying values spread over several concepts.

$\mathcal{T}$:Employee($hasSsn$, $hasName$, $hasAddress$, $hasAge$) $\rightarrow$

$\mathcal{O}$:Employee(y), $\mathcal{O}$:hasSsn(y, $hasSsn$), $\mathcal{O}$:hasName(y, $hasName$),

$\mathcal{O}$:hasAddress(y, $hasAddress$), $\mathcal{O}$:hasAge(y, $hasAge$).

Its identifier is represented by

$$\text{identify}_{Employee}(y, hasSsn) \quad \rightarrow \quad \mathcal{O}\text{:Employee}(y),\ \mathcal{O}\text{:hasSsn}(y, hasSsn).$$

In turn, $\tau(Dependent)$ produces the table

$\mathcal{T}$:Dependent(deName, hasSsn, birthDate,...),

whose anchor is Dependent. Note that the hasSsn column is a foreign key referencing the hasSsn column in the $\mathcal{T}$:Employee table. Accordingly, its semantics is represented as:

$\mathcal{T}$:Dependent($deName$, $hasSsn$, $birthDate$,...) $\rightarrow$ $\mathcal{O}$:Dependent(y), $\mathcal{O}$:Employee(w),
$\mathcal{O}$:dependents_of(y, w),
identify$_{Employee}$(w, $hasSsn$),
$\mathcal{O}$:deName(y, $deName$),
$\mathcal{O}$:birthDate(y, $birthDate$) ...

and its identifier is represented as:

identify$_{Dependent}$(y, $deName$, $hasSsn$) $\rightarrow$ $\mathcal{O}$:Dependent(y), $\mathcal{O}$:Employee(w),
$\mathcal{O}$:dependents_of(y, w),
identify$_{Employee}$(w, $hasSsn$),
$\mathcal{O}$:deName(y, $deName$).

τ can be applied similarly to the other objects in Figure 4.2. $\tau(works\_for)$ produces the table

works_for(hasSsn, hasDeptNumber).

$\tau(participates)$ generates the table

participates(hasNumber, hasDeptNumber).

Please note that the anchor of the table generated by $\tau(works\_for)$ is Employee, while no single anchor is assigned to the table generated by $\tau(participates)$.

■

The second step of the er2rel schema design methodology suggests that the schema generated by τ can be modified by (repeatedly) *merging* into the table T_0 of an entity E the table T_1 of some functional relationship involving the same entity E (which has a foreign key reference to T_0). If the semantics of T_0 is $T_0(K, V) \rightarrow \phi(K, V)$, and of T_1 is $T_1(K, W) \rightarrow \psi(K, W)$, then the semantics of table T=merge(T_0,T_1) is, to a first approximation,

$$T(K, V, W) \rightarrow \phi(K, V), \psi(K, W). \tag{4.1}$$

And the anchor of T is the entity E. (We defer the description of the treatment of null values which can arise in the non-key columns of T_1 appearing in T.) For example, we could merge the table $\tau(Employee)$ with the table $\tau(works\_for)$ in Example 4.3.2 to form a new table

$\mathcal{T}$:Employee2(hasSsn, hasName, hasAddress, hasAge, hasDeptNumber),

where the column $hasDeptNumber$ is an f.k. referencing $\tau(Department)$. The semantics of the table is:

$\mathcal{T}$:Employee2($hasSsn, hasName, hasAddress, hasAge, hasDeptNumber$)→
$\mathcal{O}$:Employee(y), $\mathcal{O}$:hasSsn($y, hasSsn$), $\mathcal{O}$:hasName($y, hasName$),
$\mathcal{O}$:hasAddress($y, hasAddress$), $\mathcal{O}$:hasAge($y, hasAge$), $\mathcal{O}$:Department(w),
$\mathcal{O}$:works_for(y, w), $\mathcal{O}$:hasDeptNumber($w, hasDeptNumber$).

Please note that one conceptual model may result in several different relational schemas, since there are choices in which direction a one-to-one relationship is encoded (which entity acts as a key), and how tables are merged. Note also that the resulting schema is in Boyce-Codd Normal Form, if we assume that the only functional dependencies are those that can be deduced from the ER schema (as expressed in FOL).

In this subsection, we assume that the CM has no so-called "recursive" relationships relating an entity to itself, and no attribute of an entity corresponds to multiple columns of any table generated from the CM. (We deal with these in Section 4.3.3.) Note that by the latter assumption, we rule out for now the case when there are several relationships between a weak entity and its owner entity, such as hasMet connecting Dependent and Employee, because in this case $\tau(hasMet)$ will need columns deName,ssn1,ssn2, with ssn1 helping to identify the dependent, and ssn2 identifying the (other) employee they met.

Now we turn to the algorithm for finding the semantics of a table in terms of a given CM. It amounts to finding the semantic trees between nodes in the set $\mathcal{C}_T$ singled out by the correspondences from columns of the table T to attributes in the CM. As mentioned previously, the algorithm works in several steps:

1. Determine a skeleton tree connecting the concepts corresponding to key columns; also determine, if possible, a unique anchor for this tree.
2. Link the concepts corresponding to non-key columns using shortest functional paths to the skeleton/anchor tree.
3. Link any unaccounted-for concepts corresponding to other columns by arbitrary shortest paths to the tree.

To flesh out the above steps, we begin with the tables created by the standard design process. If a table is derived by the er2rel methodology from an ER_0 diagram, then Table 4.1 provides substantial knowledge about how to determine the skeleton tree. However, care must be taken when weak entities are involved. The following example describes a right process to discover the skeleton and the anchor of a weak entity table.

Example 4.3.3. Consider table

$\mathcal{T}$:Dept(number,univ, dean),

with foreign key (f.k.) univ referencing table

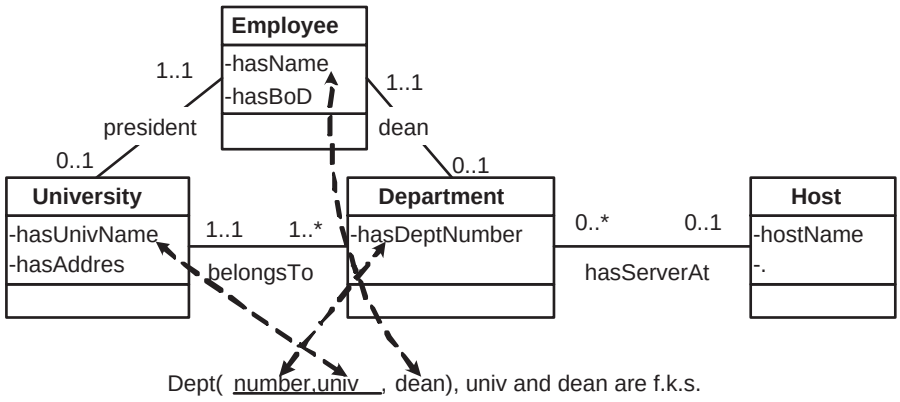

Figure 4.3: Finding Correct Skeleton Trees and Anchors.

$\mathcal{T}$:Univ(name, address)

and correspondences shown in Figure 4.3.

We can tell that $\mathcal{T}$:Dept represents a weak entity since its key has one f.k. as a subset (referring to the strong entity on which Department depends). To find the skeleton and anchor of the table $\mathcal{T}$:Dept, we first need to find the skeleton and anchor of the table referenced by the f.k. univ. The answer is University. Next, we should look for a total functional edge (path) from the correspondent of number, which is concept Department, to the anchor, University. As a result, the link `Department ---belongsTo-->- University` is returned as the skeleton, and Department is returned as the anchor. Finally, we can correctly identify the dean relationship as the remainder of the connection, rather than the president relationship, which would have seemed a superficially plausible alternative to begin with.

Furthermore, suppose we need to interpret the table

$\mathcal{T}$:Portal(dept,univ, address)

with the following correspondences:

$\mathcal{T}$:Portal.dept$\rightsquigarrow\mathcal{O}$:Department.hasDeptNumber

$\mathcal{T}$:Portal.univ$\rightsquigarrow\mathcal{O}$:University.hasUnivName

$\mathcal{T}$:Portal.address$\rightsquigarrow\mathcal{O}$:Host.hostName,

where not only is {dept,univ} the key but also an f.k. referencing the key of table $\mathcal{T}$:Dept. To find the anchor and skeleton of table $\mathcal{T}$:Portal, the algorithm first has to recursively work on the referenced table. This is also needed when the owner entity of a weak entity is itself a weak entity.

■

Figure 4.4 shows the function getSkeleton which returns a set of (skeleton, anchor)-pairs, when given a table T and a set of correspondences L from key(T). The function is essentially a recursive algorithm attempting to reverse the function τ in Table 4.1. In order to accommodate tables not designed according to er2rel, the algorithm has branches for finding minimum spanning/Steiner trees as skeletons.

In order for getSkeleton to terminate, it is necessary that there be no cycles in f.k. references in the schema. Such cycles (which may have been added to represent additional integrity constraints, such as the fact that a property is total) can be eliminated from a schema by replacing the tables involved with their outer join over the key. getSkeleton deals with strong entities and their functional relationships in step (1), with weak entities in step (2.b), and so far, with functional relationships of weak entities in (2.a). In addition to being a catch-all, step (2.c) deals with tables representing many-many relationships (which in this section have key $K = F_1F_2$), by finding anchors for the ends of the relationship, and then connecting them with paths that are not functional, even when every edge is reversed.

To find the entire semantic tree of a table T, we must connect the concepts that have attributes corresponding to the rest of the columns, i.e., nonkey(T), to the anchor(s). The connections should be (shortest) functional edges (paths), since the key determines at most one value for them; however, if such a path cannot be found, we use an arbitrary shortest path. The function getTree, shown in Figure 4.5, achieves this goal.

The following example illustrates the use of getTree when we seek to interpret a table using a different CM than the one from which it was originally derived.

Function getSkeleton(T,L)
Input: table T, correspondences L for key(T)
Output: a set of (skeleton tree, anchor) pairs
Steps:
Suppose key(T) contains f.k.s F_1,...,F_n referencing tables $T_1(K_1)$,..,$T_n(K_n)$;

1. If $n \leq 1$ and onc(key(T))[a] is just a singleton set $\{C\}$, then return $(C, \{C\})$.[b] */*T is likely about a strong entity: base case.*/*

2. Else, let L_i=$\{T_i.K_i \rightsquigarrow L(T, F_i)\}$ */*translate corresp's thru f.k. reference.*/*;
   compute (Ss_i, Anc_i) = getSkeleton(T_i, L_i), for $i = 1, .., n$.

   (a) If key(T) $= F_1$, then return (Ss_1, Anc_1). */*T looks like the table for the functional relationship of a weak entity, other than its identifying relationship.*/*

   (b) If key(T)=F_1A, where columns A are not part of an f.k. then */*T is possibly a weak entity*/*
   if $Anc_1 = \{N_1\}$ and onc(A) $= \{N\}$ such that there is a (shortest) total functional path π from N to N_1, then return (combine[c](π, Ss_1), $\{N\}$). */*N is a weak entity. cf. Example 4.3.3.*/*

   (c) Else suppose key(T) has non-f.k. columns $A[1], \ldots A[m]$, ($m \geq 0$); let N_s=$\{Anc_i, i = 1, .., n\} \cup \{onc(A[j]), j = 1, .., m\}$; find skeleton tree S' connecting the nodes in N_s where any pair of nodes in N_s is connected by a (shortest) non-functional path; return (combine(S', $\{Ss_j\}$), N_s). */*Deal with many-to-many binary relationships; also the default action for non-standard cases, such as when not finding identifying relationship from a weak entity to the supposed owner entity. In this case no unique anchor exists.*/*

[a] onc(X) is the function which gets the set M of concepts having attributes corresponding to the columns X.

[b] Both here and elsewhere, when a concept C is added to a tree, so are edges and nodes for C's attributes that appear in L.

[c] Function combine merges edges of trees into a larger tree.

Figure 4.4: The getSkeleton Function

Example 4.3.4. In Figure 4.6, the table

$\mathcal{T}$:Assignment(emp,proj, site)

was originally derived from a CM with the entity Assignment shown on the right-hand side of the vertical dashed line. To interpret it by the CM on the left-hand side, the function getSkeleton, in Step 2.c, returns Employee ---assignedTo--- Project as the skeleton, and no single anchor exists. The set {Employee, Project} accompanying the skeleton is returned. Subsequently,

Function getTree(T,L)
Input: table T, correspondences L for columns(T)
Output: set of semantic trees [a]
Steps:

1. Let L_k be the subset of L containing correspondences from key(T); compute (S', Anc')=getSkeleton(T,L_k).
2. If onc(nonkey(T)) $-$ onc(key(T)) is empty, then return (S', Anc'). */*if all columns correspond to the same set of concepts as the key does, then return the skeleton tree.*/*
3. For each f.k. F_i in nonkey(T) referencing $T_i(K_i)$:
   let $L_k^i = \{T_i.K_i \rightsquigarrow L(T, F_i)\}$, and compute (Ss_i'', Anc_i'')= getSkeleton(T_i,L_k^i). */*recall that the function $L(T, F_i)$ is derived from a correspondence $L(T, F_i, D, f, N_{f,D})$ such that it gives a concept D and its attribute f ($N_{f,D}$ is the attribute node in the CM graph.)*/*
   find π_i=shortest functional path from Anc' to Anc_i''; let S = combine($S', \pi_i, \{Ss_i''\}$).
4. For each column c in nonkey(T) that is not part of an f.k., let N = onc(c); find π=shortest functional path from Anc' to N; update S := combine(S, π). */*cf. Example 4.3.4.*/*
5. In all cases above asking for functional paths, use a shortest path if a functional one does not exist.
6. Return S.

[a] To make the description simpler, at times we will not explicitly account for the possibility of multiple answers. Every function is extended to set arguments by element-wise application of the function to set members.

Figure 4.5: The getTree Function

the function getTree seeks for the shortest functional link from elements in {Employee, Project} to Worksite at Step 4. Consequently, it connects Worksite to Employee via works_on to build the final semantic tree.

■

To get the logic formula from a tree based on correspondence L, we provide the procedure encodeTree(S, L) shown in Figure 4.7, which basically assigns variables to nodes, and connects them using edge labels as predicates.

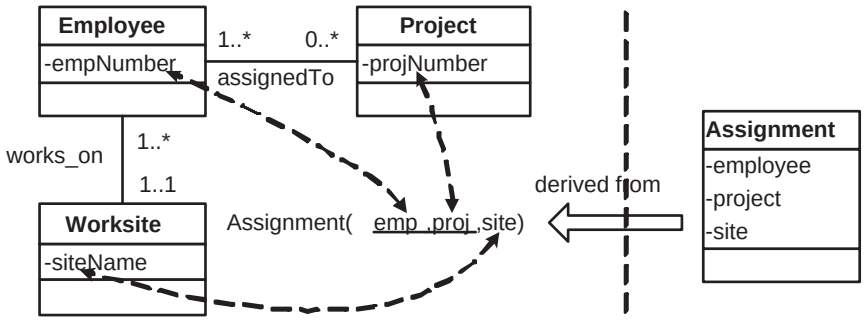

Figure 4.6: Independently Developed Table and CM.

Function encodeTree(S,L)
Input: subtree S of CM graph, correspondences L from table columns to attributes of concept nodes in S.
Output: variable name generated for root of S, and conjunctive formula for the tree.
Steps: Suppose N is the root of S. Let $\Psi = true$.

1. If N is an attribute node with label f
   - find d such that $L(\_, d, \_, f, N) = true$;
   - return$(d, true)$. */*for leaves of the tree, which are attribute nodes, return the corresponding column name as the variable and the formula $true$.*/*
2. If N is a concept node with label C, then introduce new variable x; add conjunct $C(x)$ to Ψ; for each edge p_i from N to N_i */*recursively get the subformulas.*/*
   - let S_i be the subtree rooted at N_i,
   - let $(v_i, \phi_i(Z_i))$=encodeTree(S_i, L),
   - add conjuncts $p_i(x, v_i) \wedge \phi_i(Z_i)$ to Ψ;
3. Return (x, Ψ).

Figure 4.7: The encodeTree Function

Example 4.3.5. Figure 4.8 is the fully specified semantic tree returned by the algorithm for the table

$\mathcal{T}$:Dept(number,univ, dean)

in Example 4.3.3. Taking Department as the root of the tree, function encodeTree generates the following formula:

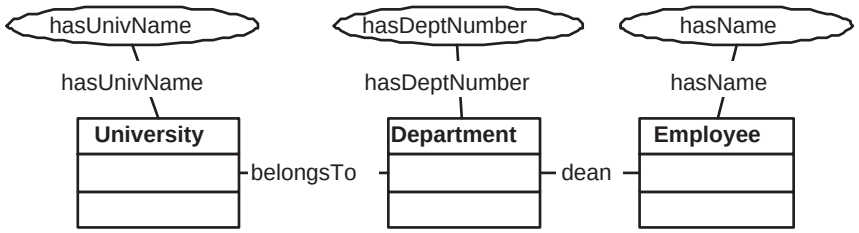

Figure 4.8: Semantic Tree For $Dept$ Table.

Department(x), hasDeptNumber($x, number$), belongsTo(x, v_1),
University(v_1), hasUnivName($v_1, univ$), dean(x, v_2),
Employee(v_2), hasName($v_2, dean$).

As expected, the formula is the semantics of the table $\mathcal{T}$:Dept as assigned by the er2rel design τ.

■

Now we turn to the properties of the mapping discovery algorithm. In order to be able to make guarantees, we have to limit ourselves to "standard" relational schemas, since otherwise the algorithm cannot possibly guess the intended meaning of an arbitrary table. For this reason, let us consider only schemas generated by the er2rel methodology from a CM encoding an ER diagram. We are interested in two properties. The first property is a sense of "completeness": the algorithm finds the correct semantics (as specified in Table 4.1). The second property is a sense of "soundness": if for such a table there are multiple semantic trees returned by the algorithm, then each of the trees would produce an indistinguishable relational table according to the er2rel mapping. (Note that multiple semantic trees are bound to arise when there are several relationships between 2 entities which cannot be distinguished semantically in a way which is apparent in the table (e.g., 2 or more functional properties from A to B). To formally specify the properties, we have the following definitions.

A *homomorphism* h from the columns of a table T_1 to the columns of a table T_2 is a one-to-one mapping h: columns(T_1)→columns(T_2), such that (i) $h(c) \in$ key(T_2) for every $c \in$ key(T_1); (ii) by convention, for a set of columns F, $h(F[1]F[2]\ldots)$ is $h(F[1])h(F[2])\ldots$; (iii) $h(Y)$ is an f.k.

of T_2 for every Y which is an f.k. of T_1; and (iv) if Y is an f.k. of T_1, then there is a homomorphism from the key(T_1') of T_1' referenced by Y to the key(T_2') of T_2' referenced by $h(Y)$ in T_2.

Definition 4.3.1. A relational table T_1 is isomorphic to another relational table T_2, if there is a homomorphism from columns(T_1) to columns(T_2) and vice versa.

Informally, two tables are isomorphic if there is a bijection between their columns which preserves recursively the key and foreign key structures. These structures have direct connections with the structures of the ER diagrams from which the tables were derived. Since the er2rel mapping τ may generate the "same" table when applied to different ER diagrams (considering attribute/column names have been handled by correspondences), a mapping discovery algorithm with "good" properties should report all and only those ER diagrams.

To specify the properties of the algorithm, suppose that the correspondence L_{id} is the identity mapping from table columns to attribute names, as set up in Table 4.1. The following lemma states the interesting property of getSkeleton.

Lemma 4.3.1. *Let CM graph $\mathcal{G}$ encode an ER$_0$ diagram $\mathcal{E}$. Let $T = \tau(C)$ be a relational table derived from an object C in $\mathcal{E}$ according to the* er2rel *rules in Table 4.1. Given L_{id} from T to $\mathcal{G}$, and L' = the restriction of L_{id} to* key*(T), then* getSkeleton*(T, L') returns (S, Anc) such that,*

- *Anc is the* anchor *of T (*anchor*(T)).*
- *If C corresponds to a (strong or weak) entity, then* encodeTree*(S, L') is logically equivalent to* identify$_C$.

Proof. The lemma is proven by using induction on the number of applications of the function getSkeleton resulting from a single call on the table T.

At the base case, step 1 of getSkeleton indicates that key(T) links to a single concept in $\mathcal{G}$. According to the er2rel design, table T is derived either from a strong entity or a functional relationship from a strong entity. For either case, anchor(T) is the strong entity, and encodeTree(S, L') is logically equivalent to identify$_E$, where E is the strong entity.

For the induction hypothesis, we assume that the lemma holds for each table that is referenced by a foreign key in T.

On the induction steps, step 2.(a) identifies that table T is derived from a functional relationship from a weak entity. By the induction hypothesis, the lemma holds for the weak entity. So does it for the relationship.

Step 2.(b) identifies that T is a table representing a weak entity W with an owner entity E. Since there is only one total functional relationship from a weak entity to its owner entity, getSkeleton correctly returns the identifying relationship. By the induction hypothesis, we prove that formula encodeTree(S, L') is logically equivalent to identify$_W$.

■

We now state the desirable properties of the mapping discovery algorithm. First, getTree finds the desired semantic mapping, in the sense that

Theorem 4.3.1. *Let CM graph $\mathcal{G}$ encode an ER$_0$ diagram $\mathcal{E}$. Let table T be part of a relational schema obtained by* er2rel *derivation from $\mathcal{E}$. Given L_{id} from T to $\mathcal{G}$, then some tree S returned by* getTree(T, L_{id}) *has the property that the formula generated by* encodeTree(S, L_{id}) *is logically equivalent to the semantics assigned to T by the* er2rel *design.*

Proof. Suppose T is obtained by merging the table for a entity E with tables representing functional relationships $f_1, \ldots, f_n$, $n \geq 0$, involving the same entity.

When $n = 0$, all columns will come from E, if it is a strong entity, or from E and its owner entiti(es), whose attributes appear in key(T). In either case, step 2 of getTree will apply, returning the skeleton S. encodeTree then uses the full original correspondence to generate a formula where the attributes of E corresponding to non-key columns generate conjuncts that are added to formula identify$_E$. Following Lemma 1, it is easy to show by induction on the number of such attributes that the result is correct.

When $n > 0$, step 1 of getTree constructs a skeleton tree, which represents E by Lemma 1. Step 3 adds edges $f_1, \ldots, f_n$ from E to other entity nodes $E_1, \ldots, E_n$ returned respectively as roots of skeletons for the other foreign keys of T. Lemma 1 also shows that these translate correctly. Steps

4 and 5 cannot apply to tables generated according to er2rel design. So it only remains to note that encodeTree creates the formula for the final tree, by generating conjuncts for $f_1, \ldots, f_n$ and for the non-key attributes of E, and adding these to the formulas generated for the skeleton subtrees at $E_1, \ldots, E_n$.

This leaves tables generated from relationships in ER_0 — the cases covered in the last two rows of Table 1 — and these can be dealt with using Lemma 1.

■

Note that this result is non-trivial, since, as explained earlier, it would not be satisfied by the current Clio algorithm [PVM$^+$02], if applied blindly to $\mathcal{E}$ viewed as a relational schema with unary and binary tables. Since getTree may return multiple answers, the following converse "soundness" result is significant.

Theorem 4.3.2. *If S' is any tree returned by* getTree(T, L_{id})*, with T, L_{id}, and $\mathcal{E}$ as above in Theorem 4.3.1, then the formula returned by* encodeTree(S', L_{id}) *represents the semantics of* some *table T' derivable by* er2rel *design from $\mathcal{E}$, where T' is isomorphic to T.*

Proof. The theorem is proven by showing that each tree returned by getTree will result in table T' isomorphic to T.

For the four cases in Table 4.1, getTree will return a single semantic tree for a table derived from an entity (strong or weak), and possibly multiple semantic trees for a (functional) relationship table. Each of the semantic trees returned for a relationship table is identical to the original ER diagram in terms of the shape and the cardinality constraints. As a result, applying τ to the semantic tree generates a table isomorphic to T.

Now suppose T is a table obtained by merging the table for entity E with n tables representing functional relationships $f_1, \ldots, f_n$ from E to some n other entities. The recursive calls getTree in step 3 will return semantic trees, each of which represent functional relationships from E. As above, these would result in tables that are isomorphic to the tables derived from the original functional relationships $f_i, i = 1...n$. By the definition of the merge operation, the result of merging these will also result in a table T' which is isomorphic to T. ■

We wish to emphasize that the above algorithms has been designed to deal even with schemas not derived using er2rel from some ER diagram. An application of this was illustrated already in Example 4.3.4 Another application of this is the use of functional paths instead of just functional edges. The following example illustrates an interesting scenario in which we obtained the right result.

Example 4.3.6. Consider the following relational table

T(personName, cityName, countryName),

where the columns correspond to, respectively, attributes pname, cname, and ctrname of concepts Person, City and Country in a CM. If the CM contains a path such that `Person` `-- bornIn ->-` `City` `-- locatedIn ->-` `Country`, then the above table, which is not in 3NF and was not obtained using er2rel design (which would have required a table for City), would still get the proper semantics:

$$\begin{aligned} \mathsf{T}(personName, cityName, countryName) \quad \rightarrow \quad & \mathsf{Person}(x_1),\ \mathsf{City}(x_2), \\ & \mathsf{Country}(x_3),\ \mathsf{bornIn}(x_1,x_2), \\ & \mathsf{locatedIn}(x_2,x_3), \\ & \mathsf{pname}(x_1,personName), \\ & \mathsf{cname}(x_2,cityName), \\ & \mathsf{ctrname}(x_3,countryName). \end{aligned}$$

If, on the other hand, there was a shorter functional path from Person to Country, say an edge labeled citizenOf, then the mapping suggested would have been:

$$\begin{aligned} \mathsf{T}(personName, cityName, countryName) \quad \rightarrow \quad & \mathsf{Person}(x_1),\ \mathsf{City}(x_2), \\ & \mathsf{Country}(x_3),\ \mathsf{bornIn}(x_1,x_2\ ), \\ & \mathsf{citizenOf}(x_1,x_3),\ \ldots \end{aligned}$$

which corresponds to the er2rel design. Moreover, had citizenOf not been functional, then once again the semantics produced by the algorithm would correspond to the non-3NF interpretation, which is reasonable since the table, having only personName as key, could not store multiple country names for a person. ■

4.3.2 ER_1: Reified Relationships

It is desirable to also have n-ary relationship sets connecting entities, and to allow relationship sets to have attributes in an ER model; we label the language allowing us to model such aspects by ER_1. Unfortunately, these features are not directly supported in most CMLs, such as OWL, which only have binary relationships. Since binary decomposition of an n-ary relationship cannot be always carried out [SJ95], such notions must instead be represented by *"reified relationships"* [DP02] (we use an annotation $\diamondsuit$ to indicate the reified relationships in a diagram): concepts whose instances represent tuples, connected by so-called "roles" to the tuple elements. So, if Buys relates Person, Shop and Product, through roles buyer, source and object, then these are explicitly represented as (functional) binary associations, as in Figure 4.9. And a relationship attribute, such as when the buying occurred, becomes an attribute of the Buys concept, such as whenBought.

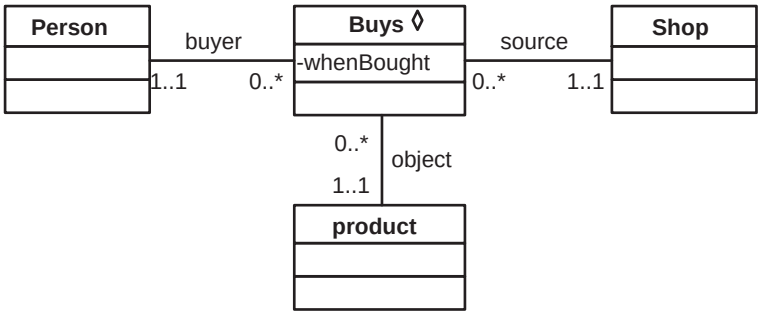

Figure 4.9: N-ary Relationship Reified.

Unfortunately, reified relationships cannot be distinguished reliably from ordinary entities in normal CMLs based on purely formal, syntactic grounds, yet they need to be treated in special ways during semantic recovery. For this reason we assume that they can be distinguished on *ontological grounds*. For example, in Dolce [GGM+02], they are subclasses of top-level concepts Quality and Perdurant/Event. For a reified relationship R, we use functions roles(R) and attribs(R) to retrieve the appropriate (binary) properties.

The er2rel design τ of relational tables for reified relationships is an extension of the treatment of binary relationships, and is shown in Table 4.2. As with entity keys, we are unable to capture in

ER model object O	**Relational Table** τ**(**O**)**	
Reified Relationship R	*columns:*	Z$X_1 \ldots X_n$
if there is a functional	*primary key:*	X_1
role r_1 for R	*f.k.'s:*	$X_1, \ldots, X_n$
E$_1$ --<- r$_1$ ->-- R	*anchor:*	R
--- r$_j$ ->-- E$_j$	*semantics:*	$T(ZX_1 \ldots X_n) \rightarrow R(y)$,$E_i(w_i)$, hasAttribs$(y, Z)$, $r_i(y, w_i)$,
let Z=attribs(R)		identify$_{E_i}(w_i, X_i), \ldots$
X_i=key($\tau(E_i)$)	*identifier:*	identify$_R(y, X_1) \rightarrow$ $R(y)$, $E_1(w)$, $r_1(y, w)$,
where E_i fills role r_i		identify$_{E_1}(w, X_1)$.
Reified Relationship R	*columns:*	Z$X_1 \ldots X_n$
if $r_1, \ldots, r_n$ are roles of R	*primary key:*	$X_1 \ldots X_n$
let Z=attribs(R)	*f.k.'s:*	$X_1, \ldots, X_n$
X_i=key($\tau(E_i)$)	*anchor:*	R
where E_i fills role r_i	*semantics:*	$T(ZX_1 \ldots X_n) \rightarrow R(y)$,$E_i(w_i)$, hasAttribs$(y, Z)$, $r_i(y, w_i)$,
		identify$_{E_i}(w_i, X_i), \ldots$
	identifier:	identify$_R(y, \ldots X_i \ldots) \rightarrow$ $R(y), \ldots E_i(w_i)$, $r_i(y, w_i)$,
		identify$_{E_i}(w_i, X_i)$,...

Table 4.2: er2rel Design for Reified Relationship.

CM situations where some subset of more than one roles uniquely identifies the relationship [JS96]. The er2rel design τ on ER$_1$ also admits the merge operation on tables generated by τ. Merging applies to an entity table with other tables of some functional relationships involving the same entity. In this case, the merged semantics is the same as that of merging tables obtained by applying τ to ER$_0$, with the exception that some functional relationships may be reified.

To discover the correct anchor for reified relationships and get the proper tree, we need to modify getSkeleton, by adding the the following case between steps 2(b) and 2(c).

- If key(T)=$F_1F_2 \ldots F_n$ and there exist reified relationship R with n roles $r_1, \ldots, r_n$ pointing at the singleton nodes in $Anc_1, \ldots, Anc_n$ respectively,
  - then let $S =$ combine$(\{r_j\}, \{Ss_j\})$, and return $(S, \{R\})$.

getTree should compensate for the fact that if getSkeleton finds a *reified* version of a many-many binary relationship, it will no longer look for an unreified one in step 2c. So after step 1. we add

- if key(T) is the concatenation of two foreign keys F_1F_2, and nonkey(T) is empty, compute (Ss_1,Anc_1) and (Ss_2, Anc_2) as in step 2. of getSkeleton; then find ρ=shortest many-many path connecting Anc_1 to Anc_2;
  - return (S') $\cup$ (combine(ρ, Ss_1, Ss_2))

In addition, when traversing the CM graph for finding shortest paths in both functions, we need to recalculate the lengths of paths when reified relationship nodes are present. Specifically, a path of length 2 passing through a reified relationship node should be counted as a path of length 1, because a reified binary relationship could have been eliminated, leaving a single edge.[3] Note that a semantic tree that includes a reified relationship node is valid only if all roles of the reified relationship have been included in the tree. Moreover, if the reified relation had attributes of its own, they would show up as columns in the table that are not part of any foreign key. Therefore, a filter is required at the last stage of the algorithm:

- If a reified relationship R appears in the final semantic tree, then so must all its role edges. And if one such R has as attributes the columns of the table which do not appear in foreign keys or the key, then all other candidate semantics need to be eliminated.

The previous version of getTree was set up so that with these modifications, roles and attributes to reified relationships will be found properly.

If we continue to assume that no more than one column corresponds to the same entity attribute, the previous theorems hold for ER_1 as well. To see this, consider the following two points. First, the tree identified for any table generated from a reified relationship is isomorphic to the one from which it was generated, since the foreign keys of the table identify exactly the participants in the relationship, so the only ambiguity possible is the reified relationship (root) itself. Second, if an entity E has a set of (binary) functional relationships connecting to a set of entities E_1,...,E_n, then merging the corresponding tables with $\tau(E)$ results in a table that is isomorphic to a reified relationship table, where the reified relationship has a single functional role with filler E and all other role fillers are the set of entities E_1,...,E_n.

[3] A different way of "normalizing" things would have been to reify even binary associations.

4.3.3 Replication

We next deal with the equivalent of the full ER_1 model, by allowing recursive relationships, where a single entity plays multiple roles, and the merging of tables for different functional relationships connecting the same pair of entity sets (e.g., works_for and manages). In such cases, the mapping described in Table 4.1 is not quite correct because column names would be repeated in the multiple occurrences of the foreign key. In our presentation, we will distinguish these (again, for ease of presentation) by adding superscripts as needed. For example, if entity set Person, with key ssn, is connected to itself by the likes property, then the table for likes will have schema $T[\underline{ssn^1}, \underline{ssn^2}]$.

During mapping discovery, such situations are signaled by the presence of multiple columns c and d of table T corresponding to the same attribute f of concept C. In such situations, we modify the algorithm to first make a copy C_{copy} of node C, as well as its attributes, in the CM graph. Furthermore, C_{copy} participates in all the object relations C did, so edges for this must also be added. After replication, we can set $\mathsf{onc}(c) = C$ and $\mathsf{onc}(d) = C_{copy}$, or $\mathsf{onc}(d) = C$ and $\mathsf{onc}(c) = C_{copy}$ (recall that $\mathsf{onc}(c)$ retrieves the concept corresponded to by column c in the algorithm). This ambiguity is actually required: given a CM with Person and likes as above, a table $T[\underline{ssn^1}, \underline{ssn^2}]$ could have two possible semantics: $\mathsf{likes}(ssn^1, ssn^2)$ and $\mathsf{likes}(ssn^2, ssn^1)$, the second one representing the inverse relationship, $likedBy$. The problem arises not just with recursive relationships, as illustrated by the case of a table $T[\underline{ssn}, addr^1, addr^2]$, where Person is connected by two relationships, home and office, to concept Building, which has an address attribute.

The main modification needed to the getSkeleton and getTree algorithms is that no tree should contain two or more functional edges of the form D --- p ->-- C and its replicate D --- p ->-- C_{copy}, because a function p has a single value, and hence the different columns of a tuple corresponding to it will end up having identical values: a clearly poor schema.

As far as our previous theorems, one can prove that by making copies of an entity E (say E and E_{copy}), and also replicating its attributes and participating relationships, one obtains an ER diagram from which one can generate isomorphic tables with identical semantics, according to the er2rel

mapping. This will hold true as long as the predicate used for **both** E and E_{copy} is $E(\_)$; similarly, we need to use the same predicate for the copies of the attributes and associations in which E and E_{copy} participate.

Even in this case, the second theorem may be in jeopardy if there are multiple possible "identifying relationships" for a weak entity, as illustrated by the following example.

Example 4.3.7. An educational department in a provincial government records the transfers of students between universities in its databases. A student is a weak entity depending for identification on the university in which the student is currently registered. A transfered student must have registered in another university before transferring. The table $\mathcal{T}$:Transferred(sno,univ, sname) records who are the transferred students, and their name. The table $\mathcal{T}$:previous(sno,univ, pUniv) stores the information about the previousUniv relationship. A CM is depicted in Figure 4.10. To

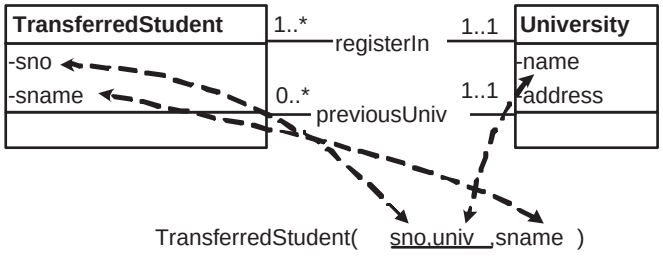

Figure 4.10: A Weak Entity and Its Owner Entity.

discover the semantics of table $\mathcal{T}$:Transferred, we link the columns to the attributes in the CM as shown in Figure 4.10. One of the skeletons returned by the algorithm for the $\mathcal{T}$:Transferred will be `TransferredStudent --- previousUniv ->-- University`. But the design resulting from this according to the er2rel mapping is not isomorphic to key($Transferred$), since previousUniv is not the identifying relationship of the weak entity TransferredStudent.

■

From above example, we can see that the problem is the inability of CMLs such as UML and OWL to fully capture notions like "weak entity" (specifically, the notion of identifying relationship), which play a crucial role in ER-based design. We expect such cases to be quite rare though – we

certainly have not encountered any in our example databases.

4.3.4 Extended ER: Adding Class Specialization

The ability to represent subclass hierarchies, such as the one in Figure 4.11 is a hallmark of CMLs and modern so-called Extended ER (EER) modeling.

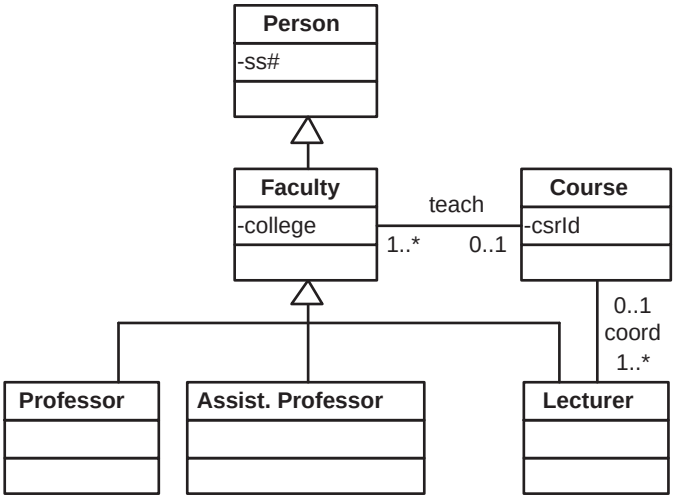

Figure 4.11: Specialization Hierarchy.

Almost all textbooks (e.g., [RG02]) describe several techniques for designing relational schemas in the presence of class hierarchies

1. Map each concept/entity into a separate table following the standard er2rel rules. This approach requires two adjustments: First, subclasses must inherit identifying attributes from a single super-class, in order to be able to generate keys for their tables. Second, in the table created for an immediate subclass C' of class C, its key $\mathsf{key}(\tau(C'))$ should also be set to reference as a foreign key $\tau(C)$, as a way of maintaining inclusion constraints dictated by the ISA relationship.

2. Expand inheritance, so that *all* attributes and relations involving a class C appear on all its subclasses C'. Then generate tables as usual for the subclasses C', though not for C itself. This approach is used only when the subclasses cover the superclass.

3. Some researchers also suggest a third possibility: "Collapse up" the information about subclasses into the table for the superclass. This can be viewed as the result of merge$(T_C, T_{C'})$, where $T_C(K, A)$ and $T_{C'}(K, B)$ are the tables generated for C and its subclass C' according to technique (1.) above. In order for this design to be "correct", the technique in [MM90] requires that $T_{C'}$ not be the target of any foreign key references (hence not have any relationships mapped to tables), and that B be non-null (so that instances of C' can be distinguished from those of C).

The use of the key for the root class, together with inheritance and the use of foreign keys to also check inclusion constraints, make many tables highly ambiguous. For example, according to the above, table $T(\underline{ss\#}, crsId)$, with $ss\#$ as the key and a foreign key referencing T', could represent at least

(a) Faculty teach Course

(b) Lecturer teach Course

(c) Lecturer coord Course.

This is made combinatorially worse by the presence of multiple and deep hierarchies (e.g., imagine a parallel Course hierarchy), and the fact that not all CM concepts are realized in the database schema, according to our scenario. For this reason, we have chosen to deal with some of the ambiguity by relying on users, during the establishment of correspondences. Specifically, the user is supposed to provide a correspondence from column c to attribute f *on the lowest class whose instances provide data appearing in the column*. Therefore, in the above example of table $T(\underline{ss\#}, crsId)$, ss# should be set to correspond to ss# on Faculty in case (a), while in cases (b) and (c) it should correspond to ss# on Lecturer. This decision was also prompted by the CM manipulation tool that we are using, which automatically expands inheritance, so that ss# appears on all subclasses.

Under these circumstances, in order to deal appropriately with designs (1.) and (2.) above, we do not need to modify our earlier algorithm in any way, as long as we first expand inheritance in the graph. So the graph would show `Lecturer -- teaches; coord ->- Course` in the

above example, and Lecturer would have all the attributes of Faculty.

To handle design (3.), we add to the graph an actual edge for the inverse of the **ISA** relation: a functional edge labeled alsoA, with lower-bound 0; e.g., `Faculty --- alsoA ->-- Lecturer`. It is then sufficient to allow in getTree for functional paths between concepts to include alsoA edges; e.g., Faculty can now be connected to Course through path alsoA followed by coord. The alsoA edge is translated into the identity predicate, and it is assigned cost zero in evaluating a functional path mixed with alsoA edge and other ordinary functional edges.[4]

In terms of the properties of the algorithm we have been considering so far, the above three paragraphs have explained that among the answers returned by the algorithm will be the correct one. On the other hand, if there are multiple results returned by the algorithm, as shown in Example 4.3.7, some semantic trees may not result in isomorphic tables to the original table, if there are more than one total functional relationships from a weak entity to its owner entity.

4.3.5 Outer Joins

As we have cautioned earlier, the definition of the semantic mapping for $T = \mathsf{merge}(T_E, T_p)$, where $T_E(K, V) \rightarrow \phi(K, V)$ and $T_p(K, W) \rightarrow \psi(K, W)$, was not quite correct. The formula

$$T(\underline{K}, V, W) \rightarrow \phi(K, V), \psi(K, W) \tag{4.2}$$

describes a join on K, rather than a left-outer join, which is what is required if p is a non-total relationship. In order to specify the equivalent of outer joins in a perspicuous manner, we will use conjuncts of the form

$$\lceil \mu(X, Y) \rceil^{Y}, \tag{4.3}$$

which will stand for the formula

$$\mu(X, Y) \ \vee \ (Y = null \wedge \neg \exists Z.\mu(X, Z)), \tag{4.4}$$

[4]It seems evident that if B ISA C, and B is associated with A via p, then this is a stronger semantic connection between C and A than if C is associated to D via a q_1, and D is associated to A via q_2.

indicating that null should be used if there are no satisfying values for the variables Y. With this notation, the proper semantics for merge is

$$T(\underline{K}, V, W) \rightarrow \phi(K, V), \lceil \psi(K, W) \rceil^{W}. \tag{4.5}$$

In order to obtain the correct formulas from trees, encodeTree needs to be modified so that when traversing a non-total edge p_i that is not part of the skeleton, in the second-to-last line of the algorithm we must allow for the possibility of v_i not existing.

4.4 Experimental Evaluation

So far, we have developed the mapping inference algorithm by investigating the connections between the semantic constraints in relational models and that in CMs. The theoretical results show that our algorithm will report the "right" semantics for most schemas designed following the widely accepted design methodology. Nonetheless, it is crucial to test the algorithm in real-world schemas and CMs to see its overall performance. To do this, we have implemented the mapping inference algorithm in our prototype system MAPONTO, and have applied it on a set of real-world schemas and CMs. In this section, we describe the implementation and provide some evidence for the effectiveness and usefulness of the prototype tool by discussing the set of experiments and our experience.

Implementation. We have implemented the MAPONTO tool as a third-party plugin of the well-known KBMS Protégé[5] which is an open platform for ontology modeling and knowledge acquisition. As OWL becomes the official ontology language of the W3C, intended for use with Semantic Web initiatives, we use OWL as the CML in the tool. This is also facilitated by the Protégé's OWL plugin [KFNM04], which can be used to edit OWL ontologies, to access reasoners for them, and to acquire instances for semantic markup. The MAPONTO plugin is implemented as a full-size user interface tab that takes advantage of the views of Protégé user interface. As shown in Figure 4.12, users can choose database schemas and ontologies, create and manipulate correspondences, gener-

[5] http://protege.stanford.edu

ate and edit candidate mapping formulas and graphical connections, and produce and save the final mappings into designated files. In addition, there is a library of other Protégé plugins that visualize ontologies graphically and manage ontology versions. Those plugins sustain our goal of providing an interactively intelligent tool to database administrators so that they may establish semantic mappings from the database to CMs more effectively.

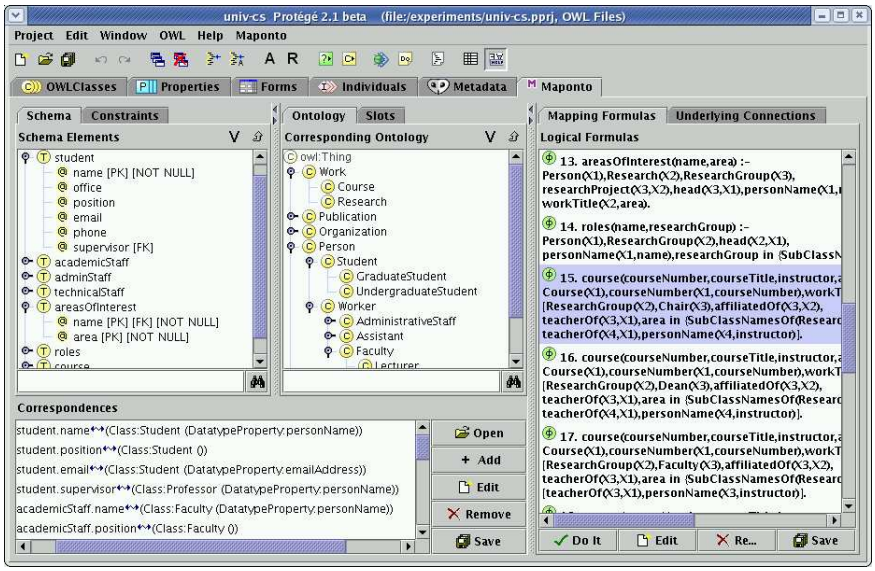

Figure 4.12: MAPONTO Plugin of Protege.

Schemas and CMs. Our test data were obtained from various sources, and we have ensured that the databases and CMs were developed independently. The test data are listed in Table 4.3. They include the following databases: the Department of Computer Science database in the University of Toronto; the VLDB conference database; the DBLP computer science bibliography database; the COUNTRY database appearing in one of reverse engineering papers [Joh94] (Although the *country* schema is not a real-world database, it appears as a complex experimental example in [Joh94], and has some reified relationship tables, so we chose it to test this aspect of our algorithm); and the test schemas in OBSERVER [MIKS96] project. For the CMs, our test data include: the academic department ontology in the DAML library; the academic conference ontology from the SchemaWeb ontology repository; the bibliography ontology in the library of the Stanford's Ontolingua server; and the CIA factbook ontology. Ontologies are described in OWL. For each ontology, the number

of links indicates the number of edges in the multi-graph resulted from object properties.

Database Schema	Number of Tables	Number of Columns	Ontology	Number of Nodes	Number of Links
UTCS Department	8	32	Academic Department	62	1913
VLDB Conference	9	38	Academic Conference	27	143
DBLP Bibliography	5	27	Bibliographic Data	75	1178
OBSERVER Project	8	115	Bibliographic Data	75	1178
Country	6	18	CIA factbook	52	125

Table 4.3: Characteristics of Schemas and CMs for the Experiments.

Results and Experience. To evaluate our tool, we sought to understand whether the tool could produce the intended mapping formula if the simple correspondences were given. We were especially concerned with the number of formulas presented by the tool for users to sift through. Further, we wanted to know whether the tool was still useful if the correct formula was not generated. In this case, we expected that a user could more easily debug a generated formula to reach the correct one instead of creating it from scratch. A summary of the experimental results are listed in Table 4.4 which shows the average size of each relational table schema in each database, the average number of candidates generated, and the average time for generating the candidates. Notice that the number of candidates is the number of semantic trees obtained by the algorithm. Also, a single edge of an semantic tree may represent the multiple edges between two nodes, collapsed using our $p; q$ abbreviation. If there are m edges in a semantic tree and each edge has n_i ($i = 1, .., m$) original edges collapsed, then there are $\prod_i^m n_i$ original semantic trees. We show below a formula generated from such a collapsed semantic tree:

$$\begin{aligned} \mathsf{TaAssignment}(courseName, studentName) \quad \rightarrow \quad & \mathsf{Course}(x_1),\ \mathsf{GraduateStudent}(x_2), \\ & \textbf{[hasTAs;takenBy]}(x_1,x_2), \\ & \mathsf{workTitle}(x_1,courseName), \\ & \mathsf{personName}(x_2,studentName). \end{aligned}$$

where, in the semantic tree, the node Course and the node GraduateStudent are connected by a single edge with label **hasTAs;takenBy**, which represents two separate edges, hasTAs and takenBy.

Database Schema	Avg. Number of Cols/per table	Avg. Number of Candidates generated	Avg. Execution time(ms)
UTCS Department	4	4	279
VLDB Conference	5	1	54
DBLP Bibliography	6	3	113
OBSERVER Project	15	2	183
Country	3	1	36

Table 4.4: Performance Summary for Generating Mappings from Relational Tables to CMs.

Table 4.4 indicates that MAPONTO only presents a few mapping formulas for users to examine. This is due in part to our compact representation of parallel edges between two nodes shown above. To measure the overall performance, we manually created the mapping formulas for all the 36 tables and compared them to the formulas generated by the tool. We observed that the tool produced correct formulas for 31 tables. This demonstrates that the tool is able to infer the semantics of many relational tables occurring in practice in terms of an independently developed CM.

We were also interested in the usefulness of the tool in those cases when the formulas generated were not the intended ones. For each such formula, we compared it to the manually generated correct one, and we wanted to know how much effort it would take to "debug" a generated formula to reach the intended one. We use the number of changes of predicate names in the formula to measure the effort. For example, for the table

Student(name, office, position, email, phone, supervisor),

the tool generated the following formula

$$\begin{aligned}&\text{Student}(x_1),\ \text{emailAddress}(x_1,email),\ \text{personName}(x_1,name),\\&\text{Professor}(x_2),\ \text{Department}(x_3),\ \text{head}(x_3,x_2),\\&\text{affiliatedOf}(x_3,x_1),\ \text{personName}(x_2,\ supervisor)...\end{aligned} \quad (1)$$

If the intended semantics for the above table columns is

$$\begin{aligned}&\text{Student}(x_1),\ \text{emailAddress}(x_1,email),\ \text{personName}(x_1,name),\\&\text{Professor}(x_2),\ \text{ResearchGroup}(x_3),\ \text{head}(x_3,x_2),\\&\text{affiliatedOf}(x_3,x_1),\ \text{personName}(x_2,\ supervisor)...\end{aligned} \quad (2)$$

then one can change the predicate Department($x3$) to ResearchGroup(x_3) in formula (1) instead of writing the entire formula (2) from scratch. Our experience working with the data sets shows that at average only about 30% of predicates in a single incorrect formula returned by the MAPONTO tool needed to be modified to reach the correct formula. This is a significant saving in terms of human labors.

Tables 4.4 indicates that execution times were not significant, since, as predicted, the search for subtrees and paths took place in a relatively small neighborhood.

We believe it is instructive to consider the various categories of problematic schemas and mappings, and the kind of future work they suggest.

(i) *Absence of tables which should be present according to* er2rel. For example, we expect the connection `Person -- researchInterest --- Research` to be returned for the table

AreaOfInterest(name, area).

However, MAPONTO returned

`Person -<- headOf --- ResearchGroup -<- researchProject --- Research`,

because there was no table for the concept Research in the schema, and so MAPONTO treated it as a weak entity table. Such problems are caused, among others, by the elimination of tables that represent finite enumerations, or ones that can be recovered by projection from tables representing total many-to-many relationships. These pose an important open problem for now.

(ii) *Mapping formula requiring selection.* The table

European(country, gnp)

means countries which are located in Europe. From the database point of view, this selects tuples representing European countries. Currently, MAPONTO is incapable of generating formulas involving the equivalent to relational selection. This particular case is an instance of the need to express "higher-order" correspondences, such as between table/column names and CM values. A similar example appears in [MHH00].

(iii) *Non-standard design.* One of the bibliography tables had a columns of author and a column

of otherAuthors for each document. MAPONTO found a formula that was close to the desired one, with conjuncts hasAuthor(d, author), hasAuthor(d, otherAuthors), but not surprisingly, could not add the requirement that otherAuthors is really the concatenation of all but the first author.

4.5 Finding GAV Mappings

Arguments have been made that the proper way to connect CMs and databases for the purpose of information integration is to show how concepts and properties in the CM can be expressed as queries over the database – the so-called GAV approach.

To illustrate the idea, consider Example 4.1.1 , from Section 4.1, where the semantic mapping we proposed was

$$\begin{aligned} \mathcal{T}\text{:Employee}(ssn, name, dept, proj) \quad \rightarrow \quad & \mathcal{O}\text{:Employee}(x_1),\ \mathcal{O}\text{:hasSsn}(x_1,ssn), \\ & \mathcal{O}\text{:hasName}(x_1,name),\ \mathcal{O}\text{:Department}(x_2), \\ & \mathcal{O}\text{:works\_for}(x_1,x_2),\ \mathcal{O}\text{:hasDeptNumber}(x_2,dept), \\ & \mathcal{O}\text{:Worksite}(x_3),\ \mathcal{O}\text{:works\_on}(x_1,x_3), \\ & \mathcal{O}\text{:hasNumber}(x_3,proj). \end{aligned}$$

In this case, we are looking for formulas which express $\mathcal{O}$:Department, $\mathcal{O}$:works_on, etc. in terms of $\mathcal{T}$:Employee, etc., as illustrated below.

We note that a strong motivation for mappings between CMs and databases expressed in this way is that they can be used to populate the CM with instances from the database – a task that is expected to be important for the Semantic Web.

An essential initial step is dealing with the fact that in the CM (as in object oriented databases), objects have intrinsic identity, which is lost in the relational data model, where this notion is replaced by external identifiers/keys. For this purpose, the standard approach is to introduce special Skolem functions that generate these identifiers from the appropriate keys, as in:

$$\mathcal{O}\text{:Employee}(\text{ff}(ssn)) \quad \rightarrow \quad \mathcal{T}\text{:Employee}(ssn,\_,\_,\_).$$

One then needs to express the external identifiers using axioms that relate these Skolem functions with the appropriate CM attributes:

$$\mathcal{O}\text{:hasSsn(ff}(ssn)\text{,}ssn) \quad \rightarrow \quad \mathcal{T}\text{:Employee}(ssn,\_,\_,\_).$$

Finally, one can express the associations by using the above identifiers:

$$\mathcal{O}\text{:works\_on(ff}(ssn)\text{,gg}(dept)) \quad \rightarrow \quad \mathcal{T}\text{:Employee}(ssn,\_,dept,\_).$$

The following less ad-hoc approach leads to almost identical results, but relies on the logical translation of the original mapping, found by the algorithms presented earlier in this paper. For example, the actual semantics of table $\mathcal{T}$:Employee is expressed by the formula

$$\begin{aligned}(\forall ssn, name, dept, proj.)\ (\mathcal{T}\text{:Employee}(ssn, name, dept, proj) \Rightarrow \\ (\exists x, y, z.)\ (\mathcal{O}\text{:Employee}(x) \wedge \mathcal{O}\text{:hasSsn}(x,ssn) \wedge \\ \mathcal{O}\text{:hasName}(x,name) \wedge \mathcal{O}\text{:Department}(y) \wedge \\ \mathcal{O}\text{:hasDeptNumber}(y,dept) \wedge \mathcal{O}\text{:works\_for}(x,y) \wedge \\ \mathcal{O}\text{:Worksite}(z) \wedge \mathcal{O}\text{:works\_on}(x,z) \wedge \mathcal{O}\text{:hasNumber}(z,proj))).\end{aligned}$$

The above formula can be Skolemized to eliminate the existential quantifiers to yield[6]:

$$\begin{aligned}(\forall ssn, name, dept.)\ (\mathcal{T}\text{:Employee}(ssn, name, dept) \Rightarrow \\ \mathcal{O}\text{:Employee(f}(ssn, name, dept)) \wedge \mathcal{O}\text{:hasSsn(f}(ssn, name, dept)\text{,}ssn) \wedge \\ \mathcal{O}\text{:hasName(f}(ssn, name, dept)\text{,}name) \wedge \\ \mathcal{O}\text{:Department(g}(ssn, name, dept)) \wedge \\ \mathcal{O}\text{:hasDeptNumber(g}(ssn, name, dept)\text{,}dept) \wedge \\ \mathcal{O}\text{:works\_for(f}(ssn, name, dept)\text{,g}(ssn, name, dept))).\end{aligned}$$

This implies logically a collection of formulas, including

[6]For simplicity, we eliminate henceforth the part dealing with projects.

$(\forall ssn, name, dept.)$ ($\mathcal{O}$:Employee(f($ssn, name, dept$)) $\Leftarrow$

$\mathcal{T}$:Employee($ssn, name, dept$)).

(($\forall ssn, name, dept.$) ($\mathcal{O}$:hasSsn(f($ssn, name, dept$),ssn) $\Leftarrow$

$\mathcal{T}$:Employee($ssn, name, dept$)).

$(\forall ssn, name, dept.)$ ($\mathcal{O}$:works_for(f($ssn, name, dept$),g($ssn, name, dept$)) $\Leftarrow$

$\mathcal{T}$:Employee($ssn, name, dept$)).

Note however that different tables, such as

$\mathcal{T}$:manages(ssn,dept)

say, introduce different Skolem functions, as in :

$\mathcal{O}$:Employee(h($ssn, dept$)) $\Leftarrow$ $\mathcal{T}$:manages($ssn, dept$).

$\mathcal{O}$:hasSsn(h($ssn, dept$),ssn) $\Leftarrow$ $\mathcal{T}$:manages($ssn, dept$).

Unfortunately, this appears to leave open the problem of connecting the CM individuals obtained from $\mathcal{T}$:manages and $\mathcal{T}$:Employee. The answer is provided by the fact that $\mathcal{O}$:hasSsn is inverse functional (ssn is a key), which means that there should be a CM axiom

$(\forall u, v, ssn.)$ ($\mathcal{O}$:hasSsn(u, ssn) $\wedge$ $\mathcal{O}$:hasSsn(v, ssn) $\Rightarrow u = v$).

This implies, among others, that

$(\forall ssn, name, dept.)$ (f($ssn, name, dept$) = h($ssn, dept$)).

So we need to answer queries over the CM using all such axioms.

A final, important connection to make in this case is with the research on answering queries using views [Hal01]: The semantic mappings found by the earlier algorithms in this paper can be regarded as view definitions for each relational tables, using conjunctive queries over CM predicates ("tables"). What we are seeking in this section is answers to queries phrased in terms of the CM predicates, but rephrased in terms of relational tables, where the data instances reside — which is exactly the problem of query answering using views. The kind of rules we proposed earlier in this

section are known as “inverse rules” [Qia96], and in fact Duschka and Levy [DGL00] even deal (implicitly) with the alias problem we mentioned above by their solution to the query answering problem in the presence of functional dependencies: keys functionally determine the rest of the columns in the table.

The one difference in our case worth noting is that we are willing to countenance answers which contain Skolem functions (since this is how we generate object id’s in the CM).

4.6 Discussion

We now discuss the limitations of our solution to the problem of discovering semantic mappings from relational schemas to CMs. First, the solution essentially infers semantic mappings from simple correspondences. To prove the “correctness” of the solution for the limited class of problems, we assume that the correspondences are correct, meaning that the column names in the tables correspond to the attributes of the concepts in the CM from which the column names were derived by the er2rel methodology. Since automatic schema matching tools often generate inaccurate and excessive correspondences, if a schema mapping tool is used for generating correspondences, it would be better to adjust the correspondences before starting the semantic mapping discovery process.

Second, correspondences are assumed to be simple and one-to-one functions from column names in tables to attributes of concepts in a CM. Nevertheless, sophisticated users may want to specify complex correspondences and let the tool to do the rest job. We leave this as a future work to be investigated. Third, in regard to our definition of semantic mapping between models, we seek for “semantically similar” associations in a schema and in a CM for discovering mapping. The guidance provided by the standard database design principles is appropriate for our situation. Sometimes, however, people may want to establish mappings that are not necessarily “semantically similar” as defined in this dissertation. We did not design our solution to deal with this situation.

Fourth, much the semantics of a database schema is encoded in a variety of constructs in the schema. For example, the data types of columns often imply some meaning to a schema. Our solution, however, does not take advantage of data types and element names in a schema, nor of

data types and names of objects (i.e., concept, attribute, and relationship) in a CM. We assume that there are implicit functions for converting data values in the domains of a schema to data values in the domains of a CM. Finally, we have limited our mapping formulas to the LAV (or conversely, GAV) formalism. A more flexible and general formalism would be global-local-as-view (GLAV), meaning a mapping formula would associate a conjunctive formula over the schema with a conjunctive formula over the CM. This might be possible if the CM would be better for interpreting a view defined over the schema. Our solution, however, do not generate GLAV mapping formula at this stage.

We envision a number of future research directions. Numerous additional sources of knowledge, including richer CMs, actual data stored in tables, linguistic and semantic relationships between identifiers in tables and a CM, can be used to refine the suggestions of MAPONTO, including providing a rank ordering for them. As in the original Clio system, more complex correspondences (e.g., from columns to sets of attribute names or class names), should also be investigated in order to generate the full range of mappings encountered in practice.

4.7 Summary

A semantic mapping relating a relational schema to a CM is a key component of many data integration systems. Such a mapping also provides precise meaning for data on the Semantic Web. In this chapter we studied a tool that assists the user in specifying a semantic mapping from a relational schema to a CM. As in many mapping discovery tools, we assume the user provides a set of simple element correspondences between the relational schema and the CM as an additional input. The tool generates a set of logic formulas each of which associates a relational table with a conjunctive formula over the CM.

Our algorithm relies on information from the database schema (key and foreign key structure) and the CM (cardinality restrictions, **ISA** hierarchies). Theoretically, our algorithm infers all and only the relevant semantics if a relational schema was generated using standard database design principles. In practice, our experience working with independently developed schemas and CMs

has shown that significant effort can be saved in specifying the LAV mapping formulas.

In next chapter, we develop a solution to the problem of discovering semantic mapping from an XML schema to a CM. We explore the semantic information encoded in the nested structure of an XML schema.

Chapter 5

Discovering Semantic Mappings from XML Schemas to CMs

In this chapter, we develop a solution for discovering semantic mappings from XML schemas to CMs. We start by reviewing XML schema and the mapping formalism. We use an example to illustrate the reason for developing a different algorithm for discovering semantic mappings for XML schemas rather than using the algorithm for relational schemas by converting XML schemas into relational schemas. Subsequently, we describe the principles and the algorithm in Section 5.2, and report the results of experimental evaluation in Section 5.3. Finally, We discuss limitations and future directions in Section 5.4, and summarize the chapter in Section 5.5.

5.1 The Problem

There is much XML data published on the current Web, since XML has become a standard format for information exchange on the Web. As a syntactic model, XML does not support integration automatically due to the heterogeneity in structures and vocabularies. In the database area, there are a number of important database problems requiring semantic mappings between XML data and CMs. These include XML data integration systems using a global conceptual schema, and peer-

to-peer data management systems [HIMT03]. Furthermore, semantic mappings between XML data and ontologies play an important part in accommodating XML data to the Semantic Web.

Mappings between XML schemas and CMs could be as simple as value correspondences between single elements or as complex as logic formulas. In almost all of the applications achieving accurate information integration, complex logic formulas are required. As for discovering semantic mapping from relational database schemas to CMs, manual creation of complex mapping formulas between XML schemas and CMs is also time-consuming and error-prone. In this chapter, we propose a solution that assists users to construct complex mapping formulas between XML schemas and CMs.

The proposed solution takes three inputs: a CM, an XML schema (actually, its unfolding into tree structures that we call *element trees*; see Section 3.1.2), and simple correspondences from XML attributes to CM datatype properties, of the kind possibly generated by already existing schema matching tools. The output is a list of complex mapping formulas possibly representing the semantics of the XML schemas in terms of the CMs.

A semantic mapping from an XML schema to a CM may consist of a set of mapping formulas each of which is from an element tree to a conjunctive formulas in the CM. An *element tree* can be constructed through doing a depth first search (DFS). A *mapping formula* between an element tree and a CM has the form $\Psi(X) \rightarrow \Phi(X, Y)$, where $\Psi(X)$ is a tree formula in the XML schema and $\Phi(X, Y)$ is a conjunctive formula in the CM.

There are several proposals in the literature for converting and storing XML data into relational databases [STH$^+$99, LC00]. A natural question is whether we could utilize the mapping algorithm we have developed for relational database schemas by converting XML schemas into relational tables. When we looked into the algorithms for converting XML schemas/DTDs into relational schemas, we noticed that the resulting relational tables were in fact the flat storage of the hierarchical data. In order to minimize the fragmentation of the hierarchical data, tuples in tables may have different meanings because they may come from different levels of the hierarchy. Their positions are maintained by pointing to different tuples generated from parents or the root in the original

hierarchy. Furthermore, each tuple is assigned a newly system-generated *id* attribute as its identifier. Consequently, applying the algorithm of discovering semantic mappings from relational tables to CMs would not produce good results because the information about keys and foreign keys, which are heavily relied on by the algorithm, become the mechanism for hierarchy maintenance. This is mainly because the goal of converting XML schemas into relational tables is to store and query XML data in relational databases. The following example illustrates the above point.

Example 5.1.1. We use the example of [STH+99] to show the point. The "hybrid inlining" technique is proposed by [STH+99] for converting XML schemas/DTDs into relational tables. The following XML DTD is given by [STH+99] for demonstrating the algorithm.

```
<!ELEMENT book (booktitle, author)>
<!ELEMENT article(title, author*,contactauthor)>
<!ELEMENT contactauthor EMPTY>
<!ATTLIST contactauthor authorID, IDREF IMPLIED>
<!ELEMENT monograph(title, author, editor)>
<!ELEMENT editor(monograph*)>
<!ATTLIST editor name CDATA #REQUIRED>
<!ELEMENT author(name, address)>
<!ATTLIST authorid ID #REQUIRED>
<!ELEMENT name(firstname?, lastname)>
<!ELEMENT firstname(#PCDATA)>
<!ELEMENT lastname(#PCDATA)>
<!ELEMENT address ANY>
```

Graphically, Figure 5.1 shows the schema graph created from the above XML DTD. A single-lined arrow "$\rightarrow$" indicates multiple occurrences of a child element, while a double-lined arrow "$\Rightarrow$" indicates single occurrence of a child element or an attribute. Subsequently, the hybrid inlining technique generates four relational tables shown in Figure 5.2, where the columns are named by the path from the root element of the table. There are several features to note in the relational tables. Each table has an ID field that serves as the key of that table. All tables corresponding to element nodes having a parent also have a parentID field that serves as a foreign key. For instance, the author table has a foreign key author.parentID that joins authors with articles.

Given a bibliographic CM containing concepts Article, Author, and a many-to-many relationship hasAuthor between Article and Author. If we try to discover the semantic mapping between the

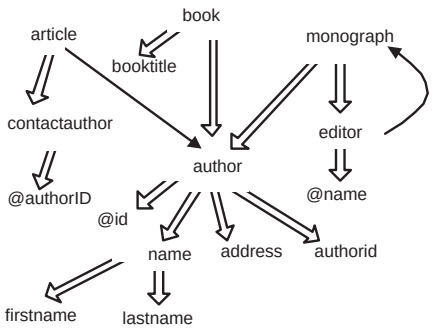

Figure 5.1: The Schema Graph Corresponding to the bibliographic DTD

book(bookID:integer, book.booktitle.isroot:boolean, book.booktitle:string,
author.name.firstname:string, author.name.lastname:string,
author.address:string, author.authorid:string).

article(articleID:integer, article.contactauthor.isroot:boolean,
article.contactauthor.authorid:string,
article.title.isroot:boolean, article.title:string.

monograph(monographID:integer, monograph.parentID:boolean,
monograph.parentCODE:integer, monograph.title:string,
monograph.editor.isroot:boolean, monograph.editor.name:string,
author.name.firstname:string, author.name.lastname:string,
author.address:string, author.authorid:string).

author(authorID:integer, author.parentID:integer, author.parentCODE:integer,
author.name.isroot:boolean, author.name.firstname.isroot:boolean,
author.name.firstname:string, author.name.lastname.isroot:boolean,
author.name.lastname:string, author.address.isroot:boolean,
author.address:string, author.authorid:string).

Figure 5.2: Relational Tables Generated by the Hybrid Inline Algorithm

tables and the CM from a set of correspondences, probably, including

$$\mathcal{T}\text{:author.authorID} \leftrightsquigarrow \mathcal{O}\text{:Author.ID},$$

$$\mathcal{T}\text{:author.author.parentID} \leftrightsquigarrow \mathcal{O}\text{:Article.ID},$$

then we would not be able to find the representation of the many-to-many relationship hasAuthor from the constraints of the relational schema. The foreign key author.parentID of the author table does not match the relationship in terms of the cardinality constraints. There are no other refer-

ential constraints representing relationships between articles and authors. On the other hand, the XML schema graph contains the parent-child edge, $article \rightarrow author$, which can be interpreted by the many-to-many relationship hasAuthor. The foreign key author.parentID actually encodes the information that an author tuple references an article tuple as a parent tuple. This information represents the hierarchical relationships between elements in the XML DTD. Consequently, the solution for discovering semantic mapping from relational schema to CM misunderstands the information encoded in the foreign key constraint.

■

From this example, we can observe that foreign key constraints sometimes are used to maintain the hierarchical information of the nested XML structure. To find the semantics of an XML schema, it is necessary to develop an algorithm that explores the XML structure directly rather than making use of referential integrity constraints of the relational schema which was obtained from the XML schema. Comparing to the XML schema mapping technique in Clio [PVM+02], we still face the challenge to connect concepts in the CM as in the relational case. Moreover, our solution will exploit the occurrence constraints specified in XML schemas, while these constraints are ignored by the Clio's algorithm.

In short, the main contributions of this chapter are as follows: (i) we propose a heuristic algorithm for finding semantic mappings, which are akin to a tree connection embedded in the CM; (ii) we enhance the algorithm by taking into account information about (a) XML Schema features such as occurrence constraints, `key` and `keyref` definitions, (b) cardinality constraints in the CM, and (c) XML document design guidelines under the hypothesis that an explicit or implicit CM existed during the process of XML document design; (iii) we adopt the accuracy metric of schema matching [MGMR02] and evaluate the tool with a number of experiments.

5.2 Mapping Discovery Algorithm

Now we turn to the algorithm for discovering semantic mapping from an element tree to a CM. The algorithm assumes a set of correspondences have been given. First, we analyze the structure of an

XML element tree to lay out several principles for the algorithm.

5.2.1 Principles

As in the relational case, we start from a methodology presented in the literature [EM01, KL01] for designing XML DTDs/schemas from a CM. We begin with basic CMs which provide constructs for concepts, attributes, and binary relationships.

Basic CMs

As with relational schemas, there is a notion of XML normal form (XNF) for evaluating the absence of redundancies and update anomalies in XML schemas [EM01]. The methodology in [EM01] claims to develop XNF-compliant XML schemas from CMs. It turns out that these "good" XML schemas are trees embedded in the graph representations of the CMs. Using the term *"element tree"* instead of *"schema tree"* in [EM01], we briefly describe the algorithm of [EM01] (called *EM-algorithm)*.

Example 5.2.1. A CM containing only binary relationships between concepts is referred to as *a "binary and canonical hypergraph"* in [EM01]. For such a CM H, the *EM-algorithm* derives an element tree T such that T is in XNF and every path of T reflects a sequence of some connected edges in H. For example, starting from the Department node of the CM in Figure 5.3 the following element tree (omitting attributes) T is obtained:

$$
\begin{aligned}
&Department[\\
&\quad (FacultyMember[\\
&\qquad (Hobby)*, (GradStudent[\\
&\qquad\quad Program, (Hobby)*])*])*],
\end{aligned}
$$

where we use [ ] to indicate hierarchy and ( )* to indicate the multiple occurrences of a child element (or non-functional edges) in element trees.

In essence, *EM-algorithm* recursively constructs the element tree T as follows: it starts from a concept node N in CM, creates tree T rooted at a node R corresponding to N, and constructs the

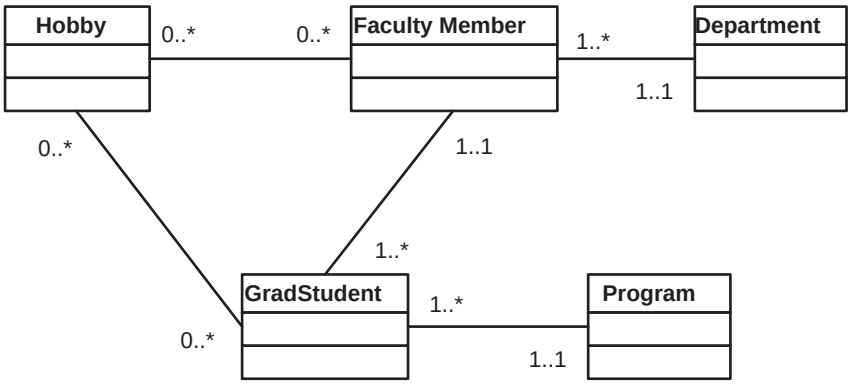

Figure 5.3: A Sample CM graph.

direct subtrees below R by following nodes and edges connected to N in CM. Finally, a largest hierarchical structure embedded within CM is identified and an edge of T reflects a semantic connection in the CM.

■

Given an XNF-compliant element tree T and the CM from which T was derived, we may assume that there is a *semantic tree* S embedded in the CM graph such that S is isomorphic to T. If the correspondences between elements in T and concepts in the CM were given, we should be able to identify S.

Example 5.2.2. Suppose elements in the element tree T of Example 5.2.1 correspond to the concepts (nodes) in Figure 5.3 by their names. Then we can recover the semantics of T recursively starting from the bottom. For the subtree T'

$$GradStudent[$$
$$Program, (Hobby)*],$$

the edge $GradStudent \Rightarrow Program$ in T' is functional and $GradStudent \rightarrow Hobby$ is non-functional. In the CM graph, we can take the concept GradStudent as the root. Then we seek for a functional edge from the concept GradStudent to the concept Program and a $1:N$ or $M:N$ edge from GradStudent to the concept Hobby. The result is the semantic tree S' consisting of two edges: `GradStudent` `--->--` `Program` and `GradStudent` `-----` `Hobby`.

Having identified S', we now move one layer up to search for a semantic tree S'' corresponding to the following subtree T''

$$FacultyMember[$$
$$(Hobby)*, (GradStudent[$$
$$Program, (Hobby)*])*].$$

The edge $FacultyMember \rightarrow Hobby$ in T'' is non-functional, and the edge from $FacultyMember$ to $GradStudent$, the root of tree S', is non-functional as well. Hence, in the CM, we build the tree S'' using the $M : N$ edge from the concept FacultyMember to the concept Hobby and the $1 : N$ edge from FacultyMember to the concept GradStudent.

Finally, we are ready to build a semantic tree S corresponding to the entire tree T

$$Department[$$
$$(FacultyMember[$$
$$(Hobby)*, (GradStudent[$$
$$Program, (Hobby)*])*])*].$$

Since we have identified a semantic tree S'' corresponding to T'', what we have to do now is to connect the concept Department to the root of S'', which is the concept FacultyMember. The connection should be a $1 : N$ or $N : M$ edge according to the occurrence constraint of the $FacultyMember$ element.

Figure 5.4 shows the final semantic tree S identified from the CM in Figure 5.3, where we use a line with arrow to indicate a functional edge. Notice that the shared concept Hobby gets duplicated in the CM graph.

■

In an element tree T, attributes are the leaves of T and often correspond to the datatype properties of concepts in a CM. Our algorithm assumes that the user specifies the correspondences from XML attributes to datatype properties in a CM, manually or using some existing schema matching tools. Given an element tree, a CM, and a set of correspondences, the algorithm attempts to identify the

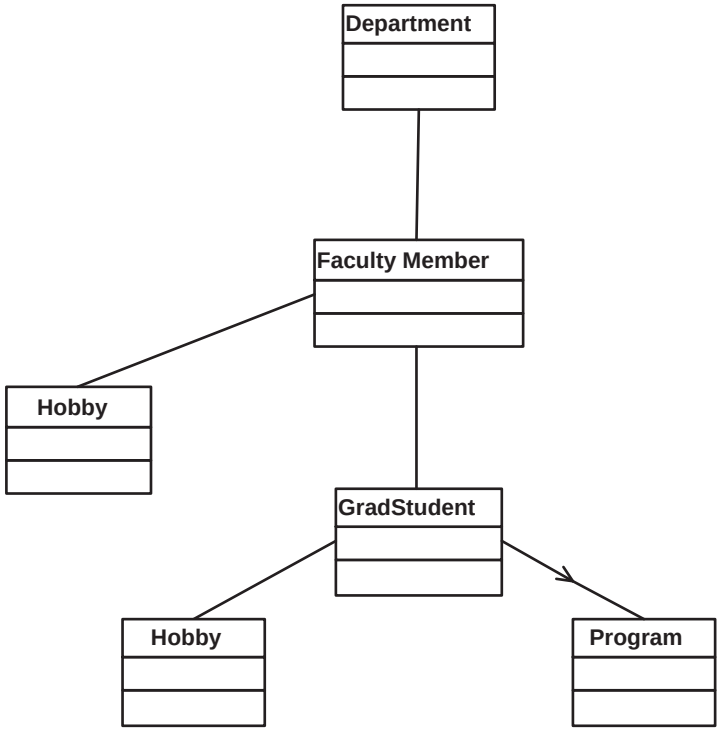

Figure 5.4: The Identified Semantic Tree

root of a semantic tree corresponding to the element tree and use "semantically matched" edges to connect the root to remaining nodes. This process is recursive and in a bottom-up fashion.

Example 5.2.3. Given an element tree T

$$GradStudent(@ln, @fn)[$$
$$Program(@pname)],$$

and a CM shown in Figure 5.5. Suppose the user specifies the following correspondences from attributes of elements to datatype properties of concepts in the CM

v_1: $\mathcal{X}$:$GradStudent.@ln \leftrightsquigarrow \mathcal{O}$:GradStudent.lastname,

v_2: $\mathcal{X}$:$GradStudent.@fn \leftrightsquigarrow \mathcal{O}$:GradStudent.firstname,

v_3: $\mathcal{X}$:$GradStudent.Program.@pname \leftrightsquigarrow \mathcal{O}$:Program.name,

where we use prefixes $\mathcal{X}$ and $\mathcal{O}$ to distinguish terms in the element tree and the CM.

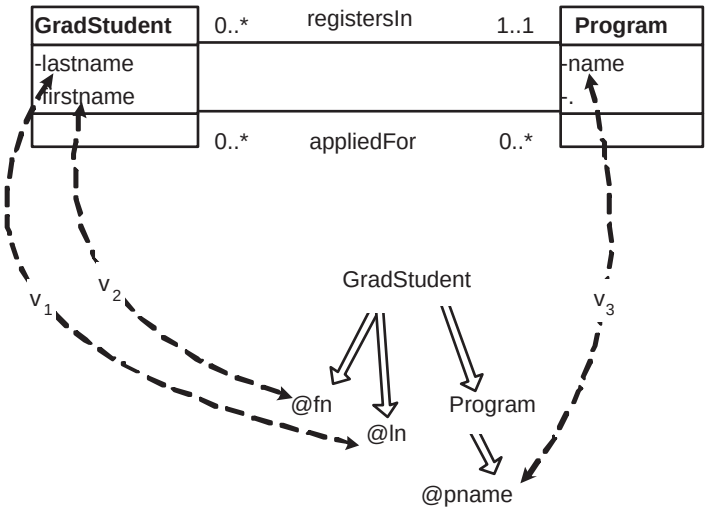

Figure 5.5: A Small CM and An element Tree

In a recursive and bottom-up fashion, we build a semantic tree S corresponding to T starting from the leaf $@pname$. The correspondence v_3 gives rise to the semantic tree S' for the leaf $@pname$, where S' is the concept Program. For the subtree $Program(@pname)$, the semantic tree is S' as well because there are no other correspondences involving the element $Program$. At this level, there are two other subtrees: $@fn$ and $@ln$. The semantic tree for both $@fn$ and $@ln$ is the concept GradStudent according to the correspondences v_1 and v_2. Let us refer to this semantic tree as S''. In connecting S'' to S', a possible solution is to assume that the root of S'' corresponds to the element tree root $GradStudent$. Therefore the connection is a functional edge from the root of S'', GradStudent, to the root of S', Program, because the connection from the element $GradStudent$ to the element $Program$ is functional (the occurrence constraint on $Program$ is 1). Consequently, we identify the semantic tree S as the connection `GradStudent --registersIn->- Program` in the CM.

■

The *first principle* of our mapping discovery algorithm is to identify the root of a semantic tree and to construct the tree recursively by connecting the root to its direct subtrees using edges in the CM graph. More precisely, for the node v_1 and its child v_2 in the element tree, if a node N_1 in the

CM is identified for the root of the semantic tree for interpreting the tree at v_1 and a node N_2 is the root of the semantic tree for the subtree at v_2, then we connect N_1 to N_2 using an edge having compatible cardinality constraints with the edge from v_1 to v_2 in the element tree.

As we may have observed, identifying the root of a semantic tree is the major obstacle. The following example illustrates the problem for an XML schema which is not XNF compliant. Such a schema can be easily encountered in reality.

Example 5.2.4. Given an element tree

$$GradStudent[$$
$$Name(@ln, @fn), Program(@pname)].$$

Suppose the user specified the following correspondences

$$\mathcal{X}{:}GradStudent.Name.@ln \leftrightsquigarrow \mathcal{O}{:}\mathsf{GradStudent.lastname},$$
$$\mathcal{X}{:}GradStudent.Name.@fn \leftrightsquigarrow \mathcal{O}{:}\mathsf{GradStudent.firstname},$$
$$\mathcal{X}{:}GradStudent.Program.@pname \leftrightsquigarrow \mathcal{O}{:}\mathsf{Program.name},$$

from the attributes of elements to the datatype properties of concepts in the CM shown in Figure 5.6.

For the element $\mathcal{X}{:}Name$ and the element $\mathcal{X}{:}Program$, we can identify two sub-trees, the concept GradStudent and the concept Program by using the correspondences. For the element $\mathcal{X}{:}GradStudent$, we have to use the two identified two sub-trees to build the final semantic tree. Since both $\mathcal{X}{:}Name$ and $\mathcal{X}{:}Program$ occur once and are at the same level, the question is which concept node is the root of the *final semantic tree*? GradStudent or Program? Since the order of nodes on the same level of the element tree does not matter, both are potential roots. Therefore, the mapping algorithm should recover functional edges from GradStudent to Program as well as from Program to GradStudent, if any.

■

This leads to the *second principle* of our algorithm. Let v_1 and v_2 be two nodes in an element tree (an element tree has element nodes and attribute nodes). Let v_2 be a child of v_1 and the maximum

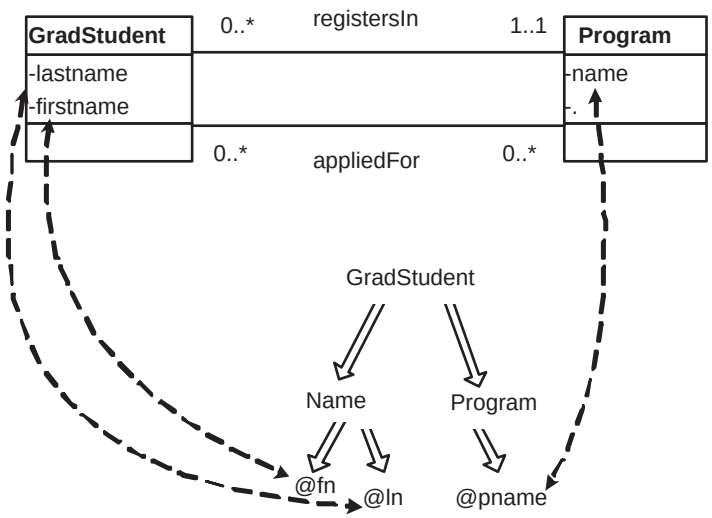

Figure 5.6: An Element Tree and A CM

occurrence constraint for v_2 is 1. For each concept C in the CM graph such that C has been identified as the root of a sub- semantic tree for the subtree at v_2, C is a potential root for building the semantic tree for the sub- element tree at v_1. If v_1 does not have a child whose maximum occurrence constraint is 1, then we find a concept node as the root of a semantic tree for the sub-element tree at v_1 as follows. The root connects to its children using non-functional paths. The tree consisting the root and its children is the minimum one if there are other trees formed by other roots connecting to the same set of children.

Unfortunately, not every functional edge from a parent node to a child node in an element tree represents a functional relationship. Specifically, some element tags are actually the collection tags. The following example illustrates the meaning of a collection tag.

Example 5.2.5. Figure 5.7 depicts an element tree and the correspondences from the element tree to a CM. The element tree and the correspondences are written in text as follows.

$GradStudent$[

$Name$(@ln, @fn), $Hobbies$[

($Hobby$(@$title$))*]]

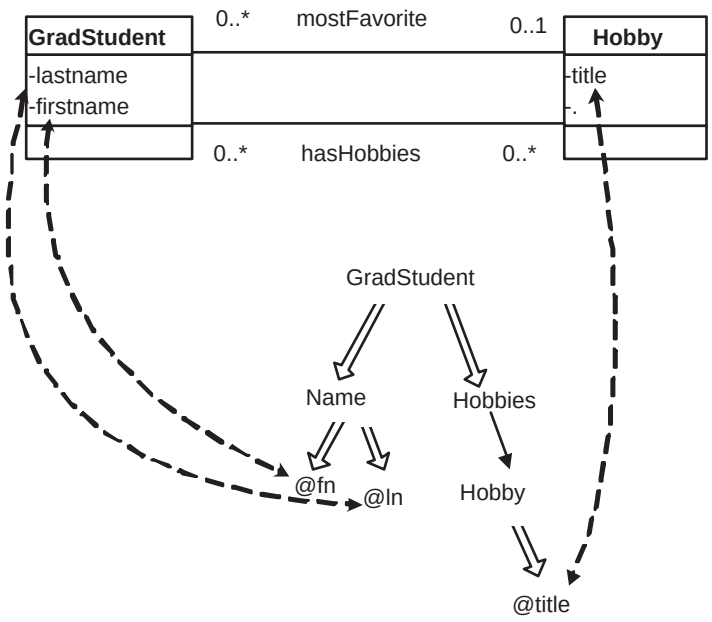

Figure 5.7: An Element Tree with a Collection Tag

$\mathcal{X}$:$GradStudent.Name.@ln \leftrightsquigarrow \mathcal{O}$:GradStudent.lastname,

$\mathcal{X}$:$GradStudent.Name.@fn \leftrightsquigarrow \mathcal{O}$:GradStudent.firstname,

$\mathcal{X}$:$GradStudent.Hobbies.Hobby.@title \leftrightsquigarrow \mathcal{O}$:Hobby.title.

The element tag $\mathcal{X}$:$Hobbies$ is a collection tag. It represents a collection of hobbies of a graduate student. Although the edge $\mathcal{X}$:$GradStudent \Rightarrow \mathcal{X}$:$Hobbies$ is functional, $\mathcal{X}$:$Hobbies \rightarrow \mathcal{X}$:$Hobby$ is non-functional. When the concept Hobby is identified as the root of the semantic tree for the subtree

$Hobbies$[

($Hobby$(@$title$))*],

Hobby should not be considered as a potential root of the semantic tree for the entire element tree.

■

Eliminating concepts corresponding to collection tags from the set of the potential roots is our *third principle*.

In most cases, we try to discover the semantic mapping between an XML schema and a CM such that they were developed independently. In such cases, we may not be able to find an isomorphic semantic tree S embedded in the CM graph. For example, for the element tree

$$City(@cityName)[$$
$$Country\ (@countryName)],$$

if a CM with a path `City -- locatedIn -->- State -- locatedIn -->- Country` is used for interpreting the element tree, the entire path is a possible answer. The *fourth principle* for discovering mappings is to find shortest paths in a CM graph instead of restricting to single edges. The composed cardinality constraints of a path should be compatible with that of the corresponding edge in the element tree.

Even though we could eliminate some collection tags from the set of potential roots to reduce the number of possible semantic trees, there may still be too many possibilities if the CM graph is large. To further reduce the size of the set of potential roots, we can make use of the `key` and `keyref` constructs in an XML schema.

Example 5.2.6. Given the element tree

$Article[$
$Title(@title), Publisher(@name), ContactAuthor(@contact), (Author(@id))*].$

If the attribute $@title$ is defined as the `key` for the element $Article$, then we should only choose the class/concept corresponding to $@title$ as the root of the semantic tree, eliminating the classes corresponding to $@name$ and $@contact$ (chosen by the second principle). Alternatively, if $@contact$ is defined as a `keyref` referencing some key, we can also eliminate the class corresponding to $@contact$. ■

So our *fifth principle* is to use `key` and `keyref` definitions to restrict the set of potential roots.

Reified Relationships

To represent n-ary relationships in the conceptual modeling language (CML), one needs to use *reified relationship (classes)* (see Section 4.3.2). For example, a CM may have class $\mathcal{O}$:Presentation

connected with functional *roles* to classes $\mathcal{O}$:Author, $\mathcal{O}$:Paper, and $\mathcal{O}$:Session, indicating participants. It is desirable to recover reified relationships and their role connections from an XML schema. Suppose the element tree

$$Presentation[$$
$$Presenter(@author), Paper(@title), Session(@eventId)],$$

represents the above ternary relationship. Then, in the CM, the root of the semantic tree is the *reified relationship class* $\mathcal{O}$:Presentation, rather than any one of the three classes which are role fillers. The *sixth principle* then is to look for *reified relationships* for element trees with only functional edges from a parent to its children that correspond to separate classes[1].

ISA Relationships

In [EM01], ISA relationships are eliminated by collapsing superclasses into their subclasses, or vice versa. If a superclass is collapsed into subclasses, correspondences can be used to distinguish the nodes in the CM. If subclasses are collapsed into their superclass, then we treat the ISA edges as special functional edges with cardinality constraints $0:1$ and $1:1$. The *last principle* is then to follow ISA edges whenever we need to construct a functional path[2].

5.2.2 Algorithm

We have presented the encodeTree(S, L) procedure, which translates a CM subtree S into a conjunctive formula, taking into account the correspondences L in Figure 4.7 of Section 4.3. The same procedure also applies to the problem of generating mapping formulas for XML schemas. The core of the solution is the following procedure constructTree(T, L) which discovers a subtree in a CM graph for an element tree after appropriately replicating nodes[3] in the CM graph.

Function constructTree(T, L)

[1] If a parent functionally connects to only two children, then it may represent an M:N binary relationship. So recover it as well.

[2] Thus, ISA is taken care of in the forthcoming algorithm by proper treatment of functional path.

[3] Replications are needed when multiple attributes correspond to the same datatype property. See Section 4.3.3 for details.

Input an element tree T, a CM graph, and correspondence L from attributes in T to datatype properties of class nodes in the CM graph.

Output set of (subtree S, root R, $collectionTag$) triples, where $collectionTag$ is a boolean value indicating whether the root corresponds to a collection tag.

Steps:

1. Suppose N is the root of tree T.
2. If N is an attribute, then find $L(\_, N, \_, \_, R) = true$; return ($\{R\}$,R, $false$). */*the base case for leaves.*/*
3. If N is an element having n edges $\{e_1, .., e_n\}$ pointing to n nodes $\{N_1, .., N_n\}$, let T_i be the subtree rooted at N_i,
   then compute (S_i,R_i, $collectionTag_i$)= constructTree(T_i, L) for $i = 1, .., n$;
   (a) If $n = 1$ and e_1 is non-functional, return (S_1,R_1, $true$);*/*N probably is a collection tag representing a set of instances each of which is an instance of the N_1 element.*/*
   (b) Else if $n = 1$ and e_1 is functional return (S_1,R_1,$collectionTag_1$).
   (c) Else if R_1=R_2=...=R_n, then return (combine$(S_1, .., S_n)$, R_1, $false$)[4].
   (d) Else let F=$\{R_{j_1}, .., R_{j_m} |$ s.t. e_{j_k} is functional and $collectionTag_{j_k} = false$ for $k = 1, .., m$, $j_k \in \{1, ..., n\}\}$ and NF=$\{R_{i_1}, .., R_{i_h} |$ s.t. e_{i_k} is non-functional, or e_{i_k} is functional and $collectionTag_{i_k} = true$ for $k = 1, .., h$, $i_k \in \{1, ..., n\}\}$, let $ans = \{\}$, */*separate nodes according to their connection types to N.*/*
      i. Try to limit the number of nodes in F by considering the following cases: 1) keep the nodes corresponding to `key` elements located on the highest level; 2) keep those nodes which do not correspond to `keyref` elements.
      ii. If $NF = \emptyset$, find a reified relationship concept R with m roles $r_{j_1}, .., r_{j_m}$ pointing to nodes in F, let S= combine($\{r_{j_k}\}$, $\{S_{j_k}\}$) for $k = 1, .., m$; let ans= $ans \cup$(S, R, $false$). If R does not exist and $m = 2$, find a non-functional shortest path p

[4]Function combine merges edges of trees into a larger tree.

connecting the two nodes R_{j_1}, R_{j_2} in F; let S= combine(p, S_{j_1}, S_{j_2}); let ans= $ans \cup$(S, R_{j_1}, $false$). */*N probably represents an n-ary relationship or many-many binary relationship (footnote of the sixth principle.)*/*

iii. Else for each $R_{j_k} \in F$ $k = 1, .., m$, find a shortest functional path p_{j_k} from R_{j_k} to each $R_{j_t} \in F \backslash R_{j_k}$ for $t = 1, .., k-1, k+1, .., m$; and find a shortest non-functional path q_{i_r} from R_{j_k} to each $R_{i_r} \in NF$ for $r = 1, .., h$; if p_{j_k} and q_{i_r} exist, let S= combine($\{p_{j_k}\}$, $\{q_{i_r}\}$,$\{S_1, .., S_n\}$); let ans=$ans \cup$(S,R_{j_k},$false$). */*pick an root and connect it to other nodes according to their connection types.*/*

iv. If $ans \neq \emptyset$, return ans; else find a minimum Steiner tree S connecting $R_1, .., R_n$, return (S,R_1, $false$). */*the default action is to find a shortest Steiner tree.*/*

It is likely that the algorithm will return too many results. Therefore, at the final stage we set a threshold N_{thresh} for limiting the number of final results presented.

5.3 Experimental Evaluation

We have implemented the mapping algorithm in the prototype tool MAPONTO and conducted a set of experiments to evaluate its effectiveness and usefulness.

Measures for mapping quality and accuracy. We first attempt to use the notions of *precision* and *recall* for the evaluation. Let R be the number of correct mapping formulas of an XML schema, let I be the number of correctly identified mapping formulas by the algorithm, and let P be the total number of mapping formulas returned. The two quantities are computed as: $precision = I/P$ and $recall = I/R$. Please note that for a single input element tree T, which has a single correct mapping formula, the algorithm either produces the formula or not. So the *recall* for T is either 0 or 1, but the *precision* may vary according to the number of output formulas. For measuring the overall quality of the mapping results, we computed the average precision and recall for all tested element trees of an XML schema.

However, precision and recall alone cannot tell us how useful the algorithm is to users. The

purpose of our tool is to *assist* users in the process of constructing complex mappings, so that productivity is enhanced. Consider the case when only one semantic mapping is returned. Even if the tool did not find the exactly right one, it could still be useful if the formula is accurate enough so that some labor is saved. To try to measure this, we adopt an accuracy metric for schema matching appearing in the literature. Consider the mapping formula $\Psi(\overline{X})\rightarrow\Phi(\overline{X},\overline{Y})$ with the formula $\Psi(\overline{X})$ encoding an element tree. The formula $\Phi(\overline{X},\overline{Y})$ encodes a semantic tree $S=(V,E)$ by using a set of unary predicates for nodes in V, a set of binary predicates for edges in E, and a set of variables, $\overline{Y}$, assigned to each node (there are predicates and variables for datatype properties as well). For a given element tree T, writing the complex mapping formula consists of identifying the semantic tree and encoding it into a conjunctive formula (which could be treated as a set of atomic predicates). Let $\Phi_1=\{a_1(\overline{Z}_1),a_2(\overline{Z}_2),..,a_m(\overline{Z}_m)\}$ encode a tree S_1, let $\Phi_2=\{b_1(\overline{Y}_1),b_2(\overline{Y}_2),..,b_n(\overline{Y}_n)\}$ encode a tree S_2. Let $D=\Phi_2\backslash\Phi_1=\{b_i(\overline{Y}_i)|$ s.t. for a given partial one-one function $f:\overline{Y}\rightarrow\overline{Z}$ representing the mapping from nodes of S_2 to nodes of S_1, $b_i(f(\overline{Y}_i))\in\Phi_1\}$. One can easily identify the mapping $f:\overline{Y}\rightarrow\overline{Z}$ by comparing the two trees S_2 and S_1 (recall a CM graph contains class nodes as well as attribute nodes representing datatype properties) so we consider that it comes for free. Let $c=|D|$. Suppose Φ_1 be the correct formula and Φ_2 be the formula returned by the tool for an element tree. To reach the correct formula Φ_1 from the formula Φ_2, one needs to delete $n-c$ predicates from Φ_2 and add $m-c$ predicates to Φ_2. On the other hand, if the user creates the formula from scratch, m additions are needed. Let us assume that additions and deletions need the same amount of effort. However, browsing the ontology for correcting formula Φ_2 to formula Φ_1 is different from creating the formula Φ_1 from scratch. So let α be a cost factor for browsing the ontology for correcting a formula, and let β be a factor for creating a formula. We define the accuracy or labor savings of the tool as $labor\ savings=1-\frac{\alpha[(n-c)+(m-c)]}{\beta m}$. Intuitively, $\alpha<\beta$, but for a worst-case bound let us assume $\alpha=\beta$ in this study. Notice that in a perfect situation, $m=n=c$ and $labor\ savings=1$.

Schemas and CMs. To evaluate the tool, we collected 9 XML schemas varying in size and nested structure. The 9 schemas come from 4 application domains, and 4 publicly available domain ontologies were obtained from the Web and the literature. Table 5.1 shows the characteristics of the

schemas and the ontologies; the column heads are self-explanatory. The *company* schema and ontology are obtained from [KL01] in order to test the principles of the mapping construction. The *conference* schema is obtained from [LC00]. *UT DB* is the schema used for describing the information of the database group at the University of Toronto. *SigmodRecord* is the schema for SIGMOD record. The rest of the schemas are obtained from the *Clio* test suite. The KA ontology, CIA factbook, and the Bibliographic-Data are all available on the Web.

XML Schema	Max Depth (DFS) in Schema Graph	# Nodes in Schema Graph	# Attributes in Schema Graph	Ontology	# Nodes	# Links
Company	6	30	17	Company	18	27
Conference	5	21	12	KA	105	4396
UT DB	6	40	20	KA	105	4396
Mondial	6	214	93	CIA factbook	52	77
DBLP 1	3	132	63	Bibliographic	75	749
DBLP 2	5	29	11	Bibliographic	75	749
SigmodRecord	3	16	7	Bibliographic	75	749
Amalgam 1	3	117	101	Bibliographic	75	749
Amalgam 2	3	81	53	Bibliographic	75	749

Table 5.1: Characteristics of Test XML Schemas and CMs

Experimental results. Our experiments are conducted on a Dell desktop with a 1.8GHZ Intel Pentium 4 CPU and 1G memory. The first observation is the efficiency. In terms of the execution times, we observed that the algorithm generated results on average in 1.4 seconds which is not significantly large, for our test data.

Figure 5.8 shows the average precision and recall measures of the 9 mapping pairs. For each pair of schema and ontology, the average precision and recall are computed as follows. For the element trees extracted from a schema graph, a set of correct mapping formulas is manually created. We then apply the algorithm on the element trees and ontologies to generate a set of formulas. Next we examine each of the generated formulas to count how many are correct and compute the average precision and recall. The overall average precision is 35% and overall average recall is 75%.

Finally, we evaluate the usefulness of the tool. The usefulness is evaluated in terms of the value of labor savings which is measured by the number of different predicates between a generated

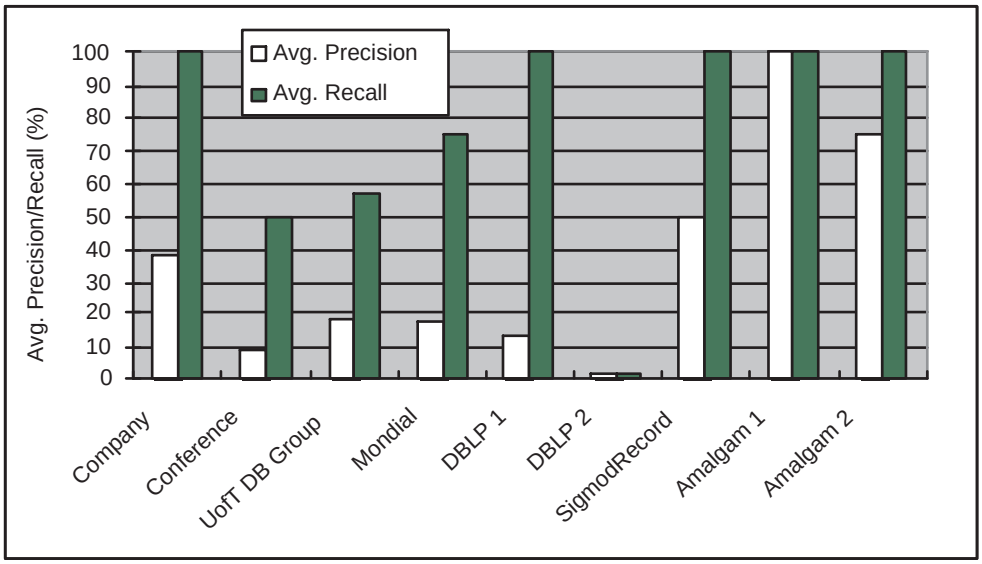

Figure 5.8: Average Recall and Precision for 9 Mapping Cases

formula and a correct formula, regardless the subject who would actually perform the correction. Figure 5.9 shows the average values of labor savings for the 9 mapping cases. For each mapping case, the average labor savings is computed as follows. Examine each incorrect formula returned by the algorithm and compute its labor saving value relative to the manually created one. Take the average value of the labor savings of all incorrect formulas. Note that even when the correct formula was identified by the algorithm, we still computed the labor savings for all incorrect ones to see how useful the tool is in case only one formula was returned. The overall average labor savings is over 80%, which is quite promising. Especially in view of the pessimistic assumption that $\alpha = \beta$ in the $labor\,savings$ formula, we take this as evidence that the tool can greatly assist users in discovering complex mappings between XML schemas and CMs with a proper schema matching tool as a front-end component.

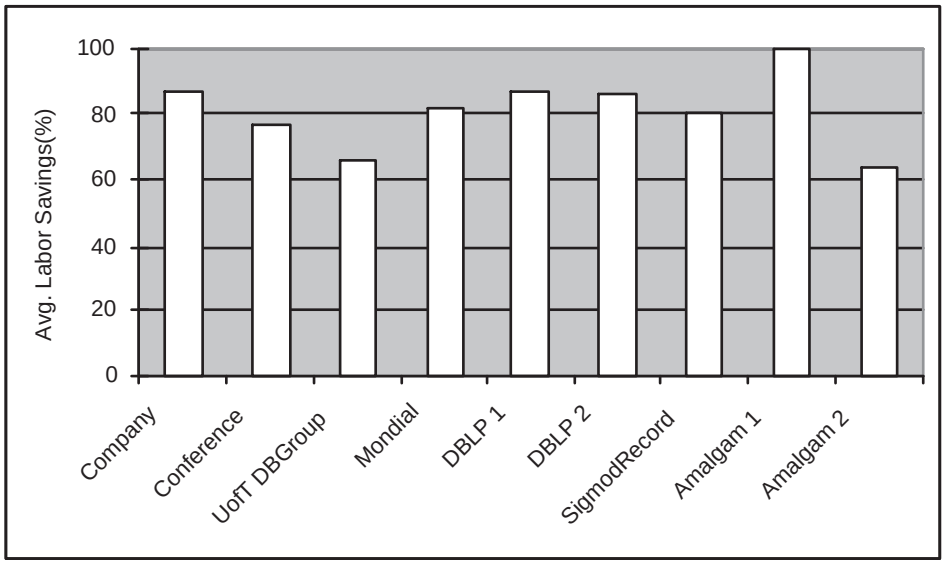

Figure 5.9: Average Labor Savings for 9 Mapping Cases

5.4 Discussion

There are several limitations in our solution for the problem of discovering semantic mappings from XML schemas to CMs. First, the proposed mapping formalism only relates a tree formula over an XML schema to a conjunctive formula over a CM. Complicated XML structures involving cycles need to be unfolded. Second, the solution assumes the user specifies a set of simple correspondences as an additional input. The correspondences are from attributes to attributes. Although the solution does not particularly deal with the correspondences specified between constructs other than attributes in both models, we believe the solution is more general and needs less user input. In fact, other types of correspondences provide more information for discovering a semantic tree. For example, if the user specified a correspondence from an element tag E to a concept, then the solution would easily use the concept as the root of a semantic tree for interpreting the element tree rooted at E. Currently, each concept corresponding to a child of E which is connected by a functional edge from E may be a potential root. Third, the solution does not explore information encoded in data instances such as XML documents. Our techniques are primarily analytical, systematically exploring information in the structures and constraints of schemas and CMs as well as database design

process.

Ontology reasoning provides a new opportunity for eliminating “unreasonable” mappings. We can encode a mapping formula into an ontology concept and reason about the ontology to find any inconsistency that may be introduced by the new encoded concept. A similar approach can be applied to eliminating “unreasonable” mappings for relational schemas as well. More details are available in [ABM06].

Integrating schema matching tools for automatically generating the correspondences and developing a filter by making use of instance data to assist users in choosing correct mappings are future investigations. Moreover, a thorough empirical usefulness study involving users with different levels of experience in schema mapping will be conducted.

5.5 Summary

In this chapter, we have looked at a new problem of discovering complex semantic mappings from XML schemas to CMs, given a set of simple correspondences from attributes to attributes. The problem is well-motivated by the increasing requirements of annotating XML documents with ontologies, translating XML data into ontologies, and integrating heterogeneous XML data sources on the semantic web. We implemented the proposed algorithm in the prototype tool, MAPONTO, for semi-automatically discovering complex mappings for users, and we evaluated the tool on a variety of real XML schemas and ontologies.

There are several novelties in our solution. First, our mapping language extends the LAV-like formalism for relational schema and relates a tree formula in an XML schema with a conjunctive formula in a CM. It subsumes the previously used formalisms [ABFS02, LS03] which deal with paths in XML tree. Second, our mapping discovery algorithm is guided by the approach for deriving “good” XML documents from conceptual models, taking various semantic information encoded in document structures into consideration. Unlike the relational schemas, hierarchical parent-child relationships and occurrence constraints are more important than key and foreign key constraints. Third, we adopted the accuracy metric of the schema matching to measure the usefulness of the tool

in real mapping cases. The experiment results show that over 80% labor could be saved.

In the next chapter, we will turn our attention to using the semantics of database schemas. We will focus on the problem of discovering mappings between relational database schemas, assuming the semantics of each schema are available in terms of some CMs.

Chapter 6

Discovering Schema Mapping Expressions Using Schema Semantics

We now address the problem of discovering schema mapping expressions by using the semantic mappings from schemas to CMs. This chapter is organized as follows. Section 6.1 describes the problem. Section 6.2 reviews the notations about schemas and semantic mappings from schemas to CMs. Section 6.3 presents motivating examples for using the semantic mappings to improve schema mapping. Section 6.4 discusses our contributions in comparison to related work. Section 6.5 describes the algorithm for schema mapping discovery. Section 6.6 evaluates using a set of experiments the proposed approach in comparison to the techniques that use only constraints in the logical schema. Finally, Section 6.7 discusses the pros and cons of our solution and points to possible future directions, and Section 6.8 summarizes the chapter.

6.1 The Problem

Schema mapping is the problem of finding a *meaningful* relationship between different database schemas. This relationship is represented in terms of logical expressions, and it is inherently difficult to automate. Just given a source and a target relational schemas, there could be numerous ways

of connecting tables in both schemas and pairing up associations to form mappings. Since the intended relationship is often buried in the head of the database designers/creators, it is almost impossible to always derive the relationship "correctly" from syntactic structures and constraints by an automaton. Therefore, interactive and semi-automatic tools are assumed to be the solution. As we have mentioned before, such a tool often employs a two-phase paradigm: first to specify the simple correspondences between schema elements such as table columns, then to derive plausible declarative mapping expressions for users to sift through.

Considering the first phase for specifying element correspondences is usually done manually or by some schema matching tools, we are interested in the problem of *deriving plausible declarative mapping expressions from element correspondences.* The element correspondences we consider will be quite simple: pairs of column names in the source and target relational schema, presumably signifying that data from the source column will contribute to data to appear in the target column. For example, in Figure 6.1, v_1 is a correspondence between column eid of table Employee in the source and the column eid of Emp in the target.

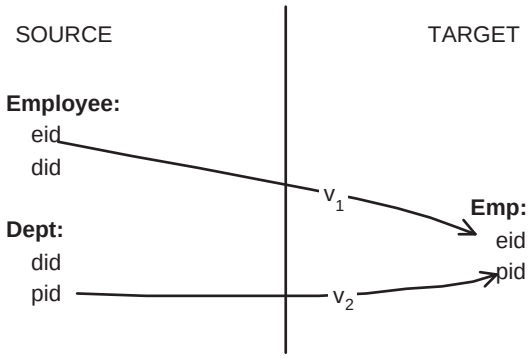

Figure 6.1: Simple Correspondences between Source and Target

Given a set of simple correspondences from a source schema to a target schema, a schema mapping solution essentially finds an association in the source among the set of elements referred to by the correspondences and an association in the target among the set of elements referred to by the same set of correspondences. In forming mappings using pairs of associations, two important questions arise: First, how to construct a "meaningful" association among a set of elements in a

schema. Second, how to match a source association to a target association in such a way that the pair gives rise to intended relationship. Recall that current solutions such as Clio [PVM+02] and MQG [KB99] rely on integrity constraints (especially referential integrity constraints) to assemble "logically connected elements" (or logical associations). These logical associations, together with the column correspondences, then give rise to mappings between the tables. The solution, however, sometimes miss important semantic connections (as shown by the motivating examples in next section), and the interpretation is primitive in the sense that a mapping is simply a pair of logical associations covering some correspondences and there lacks a strategy for ordering alternatives. This is in part because it only exploits evidence that is present in the two schemas being mapped and in part because there is no formal definition of mapping for guiding the process of matching associations.

Compared with CMs, database schemas are often *semantically impoverished*. This chapter describes an approach which leverages CMs that encode semantics of schemas to improve schema mapping. The approach explores an additional source of information, namely the *semantics* of the database schemas expressed in terms of CMs. In previous chapters, we have studied how to capture semantics of a database schema by using a CM of the domain, and a formal description relating the CM to the database schema. In addition, we observe that obtaining the semantics of a schema is not necessarily a difficult task. For example, many database schemas are developed from a conceptual model, such as an Extended Entity-Relationship diagram. Consequently, keeping the EER schema and the mapping between the EER schema and the relational schema needs limited effort. This could be quite realistic, given the proliferation of tools for managing CMs and ontologies motivated by visions of e-Services and the Semantic Web.

It is important to note that we **do not assume** that the CMs for the source and target are identical, or are connected at the semantic level, as in many data integration proposals. Instead, we rely on the element correspondences between the table columns, which have proven to be so useful for others.

6.2 Representing Semantics for Schemas

Recall that a given CM is represented in a labeled directed graph, called *CM graph*. We assume that attributes in CMs are simple and single-valued (composite and multi-valued attributes can be transformed into concepts). Note that we will deal with n-ary relationships, relationships with attributes, and so-called higher order relationships (which relate relationships themselves) in Section 6.5.3 by reifying them. We shall eventually also reify many-to-many binary relationships (ones that are not functional in either direction) since the algorithm will treat these the same way.

In Chapter 4, we studied the problem of discovering semantics for relational schemas. The semantics of a relational table is represented by a subtree in a CM graph. We have called such a subtree a *semantic tree (or s-tree)*, where columns of the table associate uniquely with attribute nodes of the s-tree. An s-tree can be encoded in conjunctive formula, where unary predicates are used for concepts and binary predicates are used for attributes and binary relationships (see the algorithm in Figure 4.7).

Example 6.2.1. Figure 6.2 shows a CM containing three concepts: Person, Book, and Bookstore. Let writes(pname, bid) and soldAt(bid, sid) be two tables in a relational schema. In terms of data, table writes(pname, bid) stores persons and the books they wrote, and table soldAt(bid, sid) records books and the stores selling the books. The columns of the tables correspond to some attributes of concepts in the CM. The semantics of writes(pname,bid) is represented by the s-tree consisting of nodes Person and Book connected by edge writes, written textually as `Person ---writes--- Book`. The s-tree is encoded in the logical formula

$$\begin{aligned} \mathcal{T}\text{:writes}(pname, bid) \quad \rightarrow \quad & \mathcal{O}\text{:Person}(x), \mathcal{O}\text{:Book}(y), \mathcal{O}\text{:writes}(x, y), \\ & \mathcal{O}\text{:pname}(x, pname), \mathcal{O}\text{:bid}(y, bid). \end{aligned}$$

where we use prefixes $\mathcal{T}$ and $\mathcal{O}$ to distinguish terms in the relational schema and the CM.

Likewise, The semantics of soldAt(bid, sid) is represented by the s-tree consisting of nodes Book and Bookstore connected by edge soldAt, written textually as `Book ---soldAt--- Bookstore`. The s-tree is encoded in the logical formula

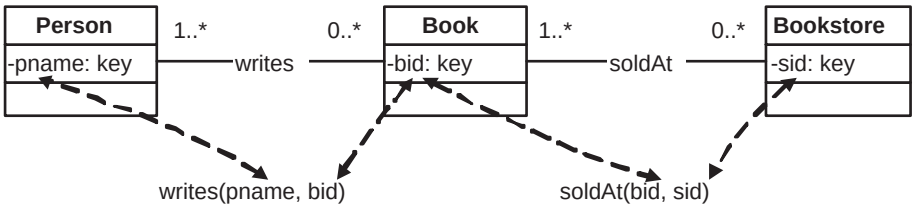

Figure 6.2: Semantics of Tables

$$\mathcal{T}\text{:soldAt}(bid, sid) \quad \rightarrow \quad \mathcal{O}\text{:Book}(x), \mathcal{O}\text{:Bookstore}(y), \mathcal{O}\text{:soldAt}(x, y), \\ \mathcal{O}\text{:bid}(x, bid), \mathcal{O}\text{:sid}(y, sid).$$

■

In order to handle multiple relationships between entities, as well as "recursive" relationships, while continuing to use trees, we duplicate concept nodes, and all the relationships they participate in (see Section 4.3.3). So, for example, the semantics of table pers(pid,name,age,spousePid) is represented by a graph with two nodes, Person and Person_{copy1}, connected by edge hasSpouse in Figure 6.3. And an additional column, pers.bestFriendPid, would require an additional node, Person_{copy2}, connected to Person by edge hasBestFriend. Note that this approach allows us to handle correctly *cyclic* RICs since the table semantics has to specify the number of times the loop has to be unfolded.

As we have noted before, there are well-known methodologies for designing logical database schemas from a CM, such as EER diagrams. We call such methodologies *er2rel designs*. We can now assert that s-trees allow the encoding of the semantics of all tables obtained by er2rel design, and there are ways of dealing with more complex formulas Φ.

Our study of discovering semantics for database schemas also associates two additional notions with the semantics of a table T: First, an *anchor*, which is the central object in the s-tree from which T is derived, if an er2rel design was used. For example, if T(c,d) was derived from a functional relationship C ---p->-- D, then C is the anchor of table T. Second, a rule expressing how classes involved in the s-tree of T are identified by columns of T. In the preceding example, class C is identified by the column c of T, while class D is identified by the column d. (More details about

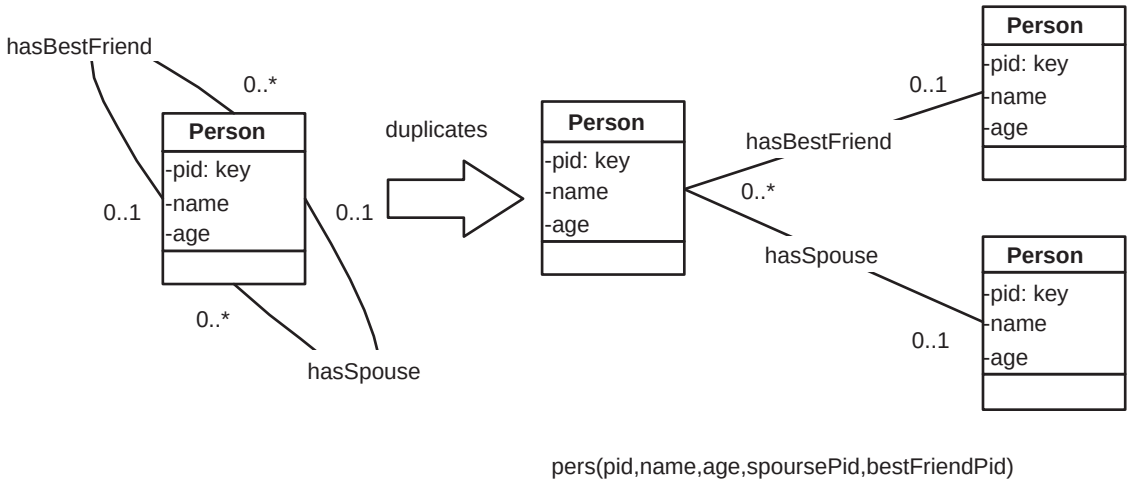

Figure 6.3: Handling Multiple and Recursive Relationships

these can be found in Section 4.3.)

6.3 Motivating Examples

Now, we highlight our motivations with illustrative (rather than exhaustive) examples.

Example 6.3.1. Consider the source relational schema given in the upper part of Figure 6.4. It contains five tables: person(pname,age), writes(pname, bid), book(bid), soldAt(bid, sid), and bookstore(sid). The underlined column name(s), such as pname, indicates the primary key of each table. A dashed arrow represents a *Referential Integrity Constraint (RIC)*, i.e., a foreign key referencing a key. (This is not required for our own algorithm, which could recover many RICs from the semantics, but is presented for comparison purposes.) For example, the dashed arrow r_1 pointing from column pname of table writes(pname,bid) to column pname of table person(pname, age), written textually as writes.pname$\subseteq$ person.pname, indicates that the values in the former column are a subset of the latter.

The semantics of the source schema is encoded by associating with each table a subgraph in the CM above it. In the CM, we take a binary relationship to link instances in its participating classes in a specific direction. For example, Person is linked to Book by writes, while Book is linked to

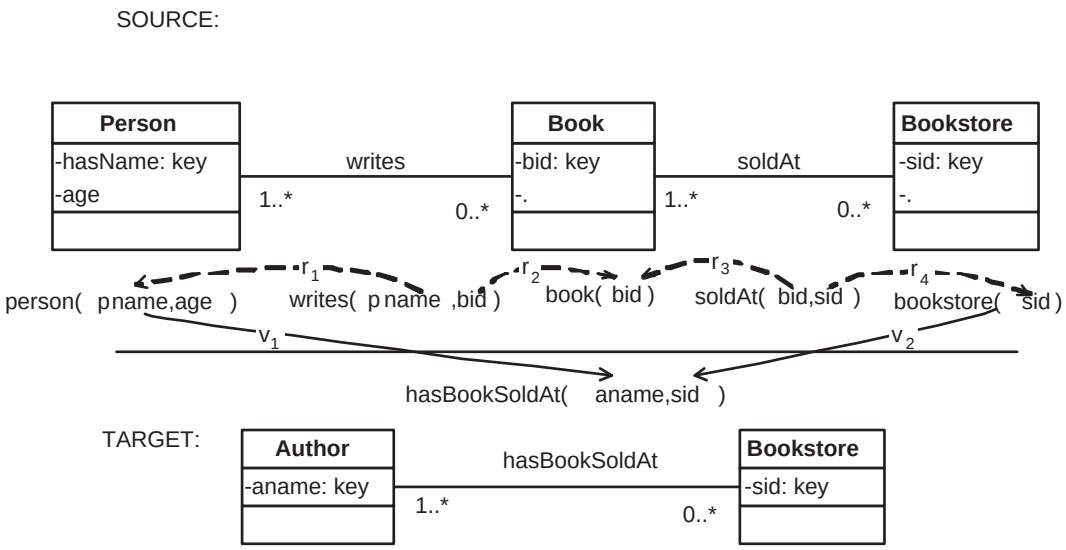

Figure 6.4: Schemas, CMs, RICs, and Correspondences

Person by its inverse, writes$^-$. To encode constraints for identifying objects, we need a special **key** annotation to indicate (collections of) attributes that act as identifiers of entities.

A *target schema* is given in the lower part of Figure 6.4. The target schema contains, among others, a table hasBookSoldAt(aname,sid). The table is associated with the CM shown below it. Now let us turn to the mapping task. To initiate the process, inter-schema correspondences need to be specified. Figure 6.4 shows two correspondences using solid lines with arrows: v_1, connecting person.pname in the source to hasBookSoldAt.aname in the target, and v_2, connecting bookstore.sid in the source to hasBookSoldAt.sid in the target. Textually, a correspondence is written as person.pname$\leftrightsquigarrow$hasBookSoldAt.aname.

Current Solution The current solutions, which we call the *RIC-based techniques*, take as input the source schema, the target schema, database constraints (including keys, foreign keys, and more generally RICs), and the correspondences. In our examples, we will use an approach proposed in Clio [PVM$^+$02], that is perhaps the most general of the solutions and generates GLAV mappings in the form of source-to-target tuple-generating dependencies (s-t tgd) [FKMP03]. Specifically, to generate a mapping expression, Clio uses an extension of the relational chase algorithm to first assemble logically connected elements into so-called *logical relations*. In this example, RICs r_1

and r_2 are applied to table writes(pname,bid) to produce the logical relation (expression):

$$S_1\text{: person(pname,age)} \bowtie \text{writes(pname, bid)} \bowtie \text{book(bid)}.$$

Likewise, we can chase the table soldAt(bid,sid) using r_3 and r_4 to produce:

$$S_2\text{: book(bid)} \bowtie \text{soldAt(bid,sid)} \bowtie \text{bookstore(sid)}.$$

In the target, a logical relation is

$$T_1\text{: hasBookSoldAt(aname,sid)}.$$

To interpret the correspondences, the RIC-based technique looks at each pair of source and target logical relations, and checks which are *covered* by the pair. For example, the pair $\langle S_1, T_1\rangle$ covers v_1, and the pair $\langle S_2, T_1\rangle$ covers v_2. So the mappings are actually written as $\langle S_1, T_1, v_1\rangle$ and $\langle S_2, T_1, v_2\rangle$. The complete algorithm will then generate the following two candidate *declarative mapping expressions* in the form of s-t tgd:

$$M_1\text{: } \forall pname, age, bid.(\text{person}(pname, age) \wedge \text{writes}(pname, bid) \wedge \text{book}(bid) \rightarrow \exists x \text{hasBookSoldAt}(pname, x)).$$

$$M_2\text{: } \forall bid, sid.(\text{book}(bid) \wedge \text{soldAt}(bid, sid) \wedge \text{bookstore}(sid) \rightarrow \exists y \text{hasBookSoldAt}(y, sid)).$$

Since, in this example, the tables person(pname) and bookstore(bid) are also logical relations, then the following are also candidate mappings:

$$M_3\text{: } \forall pname, age(\text{person}(pname, age) \rightarrow \exists x \text{hasBookSoldAt}(pname, x)).$$

$$M_4\text{: } \forall sid(\text{bookstore}(sid) \rightarrow \exists y \text{hasBookSoldAt}(y, sid)).$$

Thereafter, all candidate mappings are presented to the user for further examination and debugging.

Note that the mappings M_1 through M_4 represent incomplete data. When mappings are realized as queries (as in data exchange), Skolem functions are generally used to represent existentially quantified variables [PVM$^+$02]. In some cases, Skolem functions (and more complex mapping expressions like nested mappings) can be used to represent how data generated by different mappings

should be merged [FHH$^+$06]. However, no mapping generation algorithm that we are aware of would automatically generate a mapping that pairs authors with bookstores that stock their books, an interpretation we motivate below.

Alternate Solution We believe that the following mapping expression is more natural in this case and should be generated as a candidate:

$$M_5: \quad \forall pname, age, bid, sid.(\mathsf{person}(pname, age) \wedge \mathsf{writes}(pname, bid) \wedge \mathsf{Book}(bid) \wedge \mathsf{soldAt}(bid, sid) \wedge \mathsf{bookstore}(sid) \quad \rightarrow \quad \mathsf{hasBookSoldAt}(pname, sid)).$$

The mapping pairs in the target a person and a bookstore if the person writes a book and the book is sold at the bookstore. Looking into the semantics of the schemas, we observe that there is indeed a semantic connection between the classes Person and Bookstore, namely the composition of writes and soldAt.

Furthermore, note that the many-to-many cardinality constraint that can be inferred for the composed connection is compatible with that of the target relationship hasBookSoldAt. Contrast this to the hypothetical case when the upper bound of hasBookSoldAt would have been 1, indicating that each author is associated with at most one bookstore: we contend that such pairings are semantically incompatible, and do not lead to reasonable mapping expressions.

Note that the RIC-based techniques avoid generating lossy joins (like writes$\bowtie$soldAt), because these would provide an overabundance of logical relations, making the technique much less useful in practice. So any semantic solution must strictly limit, though not rule out, the use of such compositions.

■

Example 6.3.2. Most conceptual modeling languages support the modeling of classes connected by ISA relationships, as well as *disjointness* and *completeness* constraints concerning the subclasses.

Consider a CM, illustrated in Figure 6.5, with class Employee and two subclasses Engineer and Programmer, which are not disjoint, and cover the superclass. The bottom classes and their respective ISA relationships represent the semantics of the tables programmer(ssn,name,acnt) and engineer(ssn,name,site), forming the source schema. Suppose that the target database has

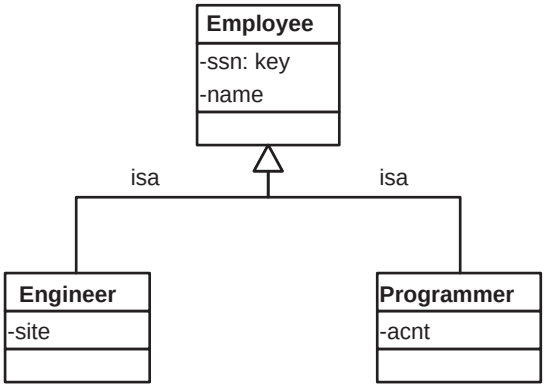

Figure 6.5: Using Rich Semantics in CM

schema employee(eid,name,site,acnt), and its CM is identical to Figure 6.5. These two databases represent alternative ways of encoding ISA hierarchies in relational tables, except for the fact that they use different identifiers, ssn and eid, as keys. Given correspondences that pair all columns with identical names (so ssn and eid do **not** correspond), the RIC-based techniques will suggest mappings ⟨programmer, employee⟩ and ⟨engineer, employee⟩, which will not merge the information about the engineer programmers. We would prefer instead a mapping that makes this connection. This will be made possible by the presence of the superclass in the CM, but absent in the database schema. ■

Example 6.3.3. In addition to cardinality considerations, the CM may contain additional information useful in eliminating or prioritizing possible mappings. For example, consider a case resembling Example 6.3.2, where information about departments and faculty are encoded using different internal keys in the source and target db. If the source had two functional relationships, chairOf and deanOf, between Department and Faculty, while the target only had one, call it foo, then even considering cardinality constraints one cannot distinguish the two mapping candidates: ⟨chairOf, foo⟩ and ⟨deanOf, foo⟩. On the other hand, if the semantics indicates that chairOf and foo are **partOf** relationships (marked by filled-in diamond in UML), but deanOf is not, then the second mapping is less likely and can be eliminated or downgraded. Because **partOf** is transitive, this would be the case even if we had to traverse a longer path of such edges in the source CM.

■

6.4 Comparisons and Contributions

We now discuss the main contributions made in this chapter in comparison to the related work. The most directly related work is obviously Clio [MHH00, PVM+02], and we have already provided some comparison of the basic techniques. Conceptual models have been used in developing graphical query interfaces for databases. A central problem is inferring a query when a user has marked some nodes in a CM diagram. We note that Zhang & Mendelzon [ZM83] applied the concept of maximal object from relational database theory to find a default connection among a set of nodes in a CM diagram. The following example shows their basic technique.

Example 6.4.1. In [ZM83], an object is defined as a relationship set together with all its participating entity sets. A maximal object is constructed by starting with an object and "growing" it into a maximal object. Objects are added to the maximal object being constructed as long as the new object joins losslessly with the objects already in the set. The cardinalities on the edges of the ER diagram is used to infer lossless joins which allow maximal objects to grow with new objects.

Figure 6.6 shows a CM diagram. Some of the maximal objects that can be inferred from the diagram are:

{Hospital ----- Lab}, (1)

{Patient --->-- Ward, Ward --->-- Hospital, Test --->-- Patient, Test --->-- Lab}, (2)

{Staff --->-- Ward, Ward --->-- Hospital}. (3)

The maximal object defined has two properties (i) the join of all the relationship and entity sets in the maximal object is lossless and (ii) the subgraph of the original CM graph corresponding to the maximal object contains no cycles. For a given set of nodes X, let T be the maximal object in the CM graph that contains every node in X and has as few nodes as possible. The set of all the T's is called *connection* on X. Queries are formed using the connection.

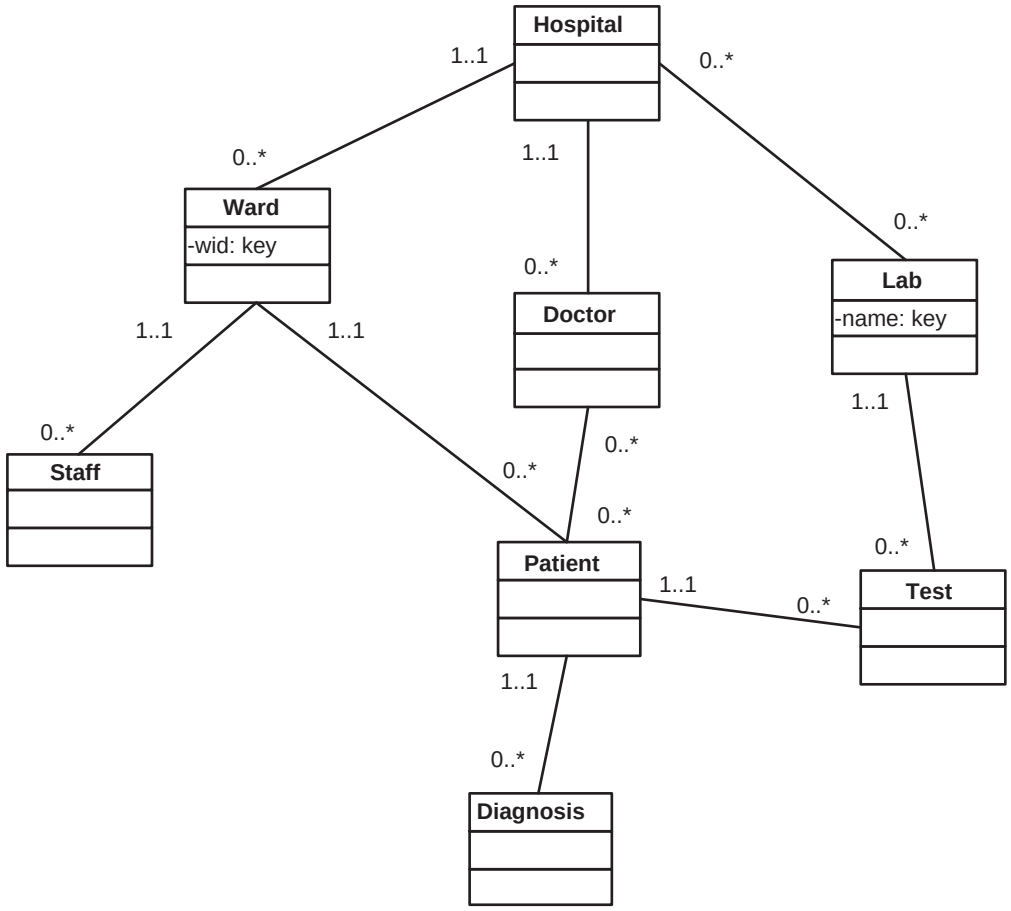

Figure 6.6: Finding Maximal Objects

Consider the query: *Find the labs associated with ward 12.* To express this query, the user can mark Lab and Ward and the output attributes. The only maximal object that contains both Lab and Ward is the second one in the above maximal object list. Therefore, the connection path is Ward – Patient – Test – Lab, and the answer will be the set of labs that have been assigned tests for patients of ward 12. Note that there are several other ways of connecting Lab to Ward. For example, the set of labs that work with the hospital where ward 12 is located. Intuitively, this connection is less "tight" than the one given by the maximal object since the join among the entity and relationship sets is lossless for the maximal object.

■

In terms of finding a meaningful association among a set of concept nodes in a CM, we also adopt the idea of lossless join. However, as Example 6.3.1 shows, sometimes the expected association may be a lossy join. Therefore, we should also consider some connections in addition to maximal objects in a CM. The following example shows how the work in [WS84] deals with the problem of connecting concept nodes in a CM by assigning different weights to different types of edges.

Example 6.4.2. In [WS84], the query inference problem is to determine the best tree containing the target graph marked by the user in a CM graph. A best query tree possesses two properties (i) it is minimal in the sense that each leaf node of the query tree is contained in the target graph and (ii) it has minimum cost. The cost of an edge is specified as follows. For an edge e between a concept node and an attribute node, the cost of e is always 1; the cost of traversing an edge e from a concept C_1 to a concept C_2 is 1 if the upper bound of the cardinality constraints on the C_2 side is 1; otherwise the cost of traversing e is μ, a sufficiently large number. Using the cost model, the query inference problem is a variation of the minimum directed cost Steiner tree problem.

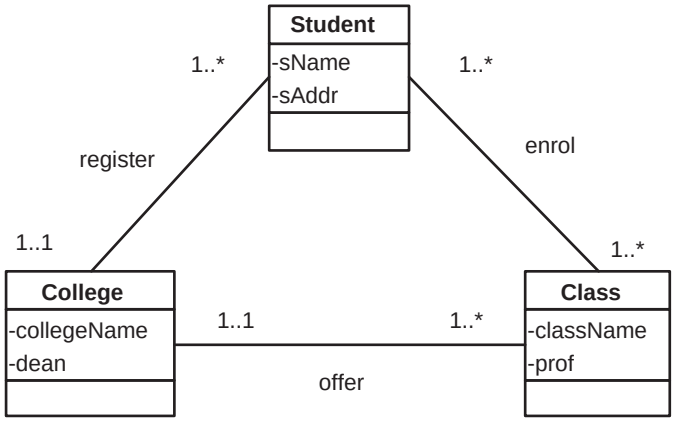

Figure 6.7: Finding Query Trees

Figure 6.7 shows a cyclic CM graph. Suppose the user has marked the target graph as the two attribute nodes sName and collegeName. There are two trees containing the target graph, namely, `Student` `--->register--` `College` and `Student` `--enrol--` `Class` `--->offer--` `College` (we have omitted the attribute nodes.) The best query tree is the first one because the cost of the second tree will add μ by traversing the edge from Student to Class. μ is sufficiently

large so that the total cost of the second tree is greater than the cost of the first tree.

■

In this example, if there is no lossless join then all edges of all connections have weight μ and these connections are still the possible connections among marked nodes; the shortest one will be returned as a result. We will also consider a connection that is a lossy join among the relationships and the concepts in the connection in our mapping discovery process. However, like above, the use of such a connection should be kept at a minimum. It would only happen when it is necessary to find a compatible connection to match one connection in other CM, as shown in Example 6.3.1 and detailed in Section 6.5.3

The contributions made in this chapter are as follows. Compared to traditional RIC-based techniques, the work presented in this chapter increases recall by, among others, slightly generalizing the use of RICs to repeatedly merging functional relationships onto the entities in the CM, where ISA is also treated as a functional relationship. It can also increase precision by eliminating candidate logical relations which cannot be consistently satisfied (e.g., because of disjointness constraints) and eliminating mappings that pair relationships with suspiciously different semantics (many-to-many with many-to-one, **partOf** with non-**partOf**). Compared to the work of using CMs for user-friendly query interfaces, the work presented in this chapter proposes a semantic solution for discovering mappings between relational schemas which is fundamentally different in that the focus is shifted from discovering meaningful connections in a *single* CM to discovering a pair of ”semantically similar” associations in different CMs. Following sections elaborate on the solution we propose.

6.5 Mapping Discovery Algorithm

In this section we study the problem of discovering a pair of matched associations in the source and target schemas. The ultimate goal is to find a pair of “semantically similar” algebraic expressions that connect the respective sets of columns linked by the correspondences. We begin with a formal description of the problem setting. Next we proceed to our basic principles for mapping discovery and use examples to illustrate the algorithm for discovering a pair of matched graphical represen-

tations in CM graphs. We present the formal algorithm after the examples. Finally, we describe a process for obtaining algebraic expressions from graphical representation.

6.5.1 Basic Criteria

The input to our algorithm consists of (i) a source relational schema $\mathcal{S}$ and a target relational schema $\mathcal{T}$, where $\mathcal{S}$ and $\mathcal{T}$ associate with their respective CMs $\mathcal{G}_S$ and $\mathcal{G}_T$ through table semantics; (ii) a set of correspondences L linking a set $L(\mathcal{S})$ of columns in $\mathcal{S}$ to a set $L(\mathcal{T})$ of columns in $\mathcal{T}$. Assuming that L specifies pairwise "similar" table columns, we seek to find a pair of "similar" algebraic expressions $\langle E_1, E_2 \rangle$ which "interpret" the correspondences L.

As shown earlier, the table semantics relate each table in the schema to an s-tree in the respective CM graph, associating with each table column a concept node in the graph through the bijective associations between columns and attribute nodes. Consequently, the set $L(\mathcal{S})$ of columns gives rise to a set $\mathcal{C}_S$ of marked concept nodes in the graph $\mathcal{G}_S$. Likewise, the set $L(\mathcal{T})$ gives rise to a set $\mathcal{C}_T$ of marked concept nodes in the graph $\mathcal{G}_T$. We call the s-trees associated with tables that have columns participating in L *pre-selected s-trees*.

Example 6.5.1. Figure 6.8 presents a source schema associated with a source CM above it and a target schema associated a target CM below it. The set of correspondences L contains two correspondences specified from the source schema to the target schema

v_1: enroll.name$\leftrightsquigarrow$taughtBy.sname

v_2: teach.instId$\leftrightsquigarrow$taughtBy.pid

$L(\mathcal{S})$ is the set {enroll.name, teach.instId}; $L(\mathcal{T})$ is the set {taughtBy.sname, taughtBy.pid}. The set $\mathcal{C}_S$ of marked concept nodes in the source CM graph contains two nodes: Student and Instructor, and the set $\mathcal{C}_T$ of marked concept nodes in the target CM graph also contains two nodes: Student and Professor. Since the source schema contains five relational tables student(name,age), course(cNum), instructor(instId), enroll(name, cNum), and teach(cNum, instId), there are five s-trees, namely, `Student`, `Course`, `Instructor`, `Student --enroll-- Course`, and `Course --taughtBy->-- Instructor`, corresponding to the five relational tables, re-

spectively. According to the correspondences v_1 and v_2, there are two pre-selected s-trees: `Student` `--enroll--` `Course` and `Course` `--taughtBy->--` `Instructor`.

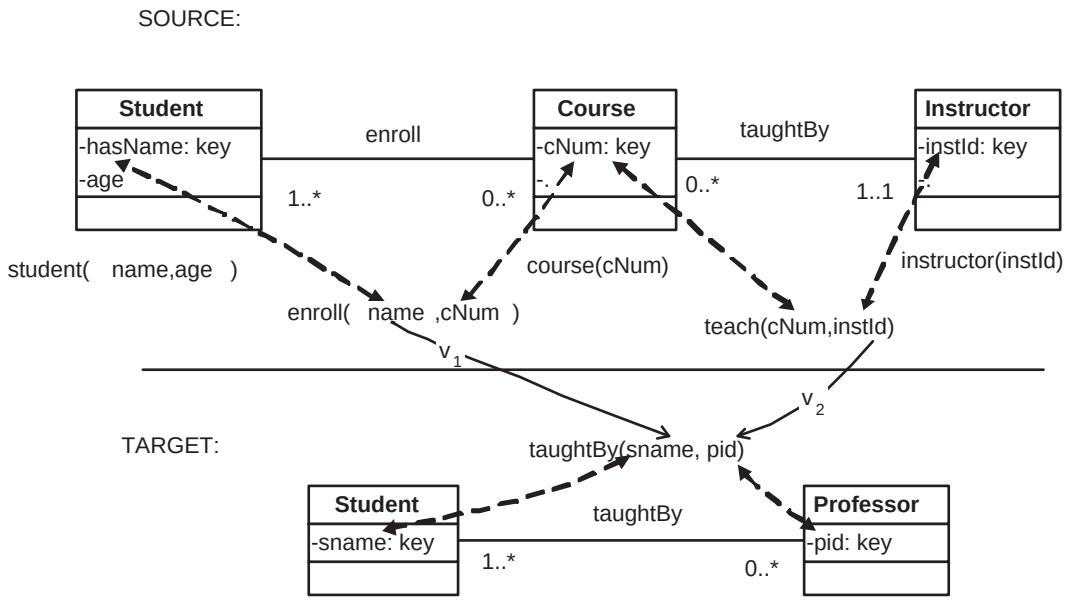

Figure 6.8: Marked Class Nodes and Pre-selected s-trees

Likewise, in the target CM graph, `Student` `--taughtBy--` `Professor` is the s-tree corresponding to the table taughtBy(sname, pid). The same s-tree is also the pre-selected s-tree in the target CM graph.

■

Our approach will consist of two major steps:

1. Find a subgraph D_1 connecting concept nodes in $\mathcal{C}_S$ and a subgraph D_2 connecting concept nodes in $\mathcal{C}_T$ such that D_1 and D_2 are semantically "similar" — we call these *conceptual subgraphs (CSG)*;

2. Translate D_1 and D_2, including the relevant attribute nodes, into algebraic expressions E_1 and E_2, and return the triple $\langle E_1, E_2, L_M \rangle$ as a mapping candidate, where $L_M \subseteq L$ is the set of correspondences covered by the pair $\langle E_1, E_2 \rangle$.

In the rest of this section, we first present the CSGs for different situations and illustrate the algorithms for discovering them using various examples. Then, we describe a process of translating a CSG into an algebraic expression by using the table semantics in terms of LAV expressions.

6.5.2 Basic Conceptual Model

We first consider basic constructs: concepts and functional binary relationships, including ISA. We delay the treatment of all other kinds of relationships to the next subsection.

There are many ways to connect the marked nodes in $\mathcal{C}_S$ to create CSGs; similarly for $\mathcal{C}_T$. We propose to systematically explore the information encoded in the correspondences and the table semantics to discover a pair of similar CSGs. First, a node $v \in \mathcal{C}_S$ corresponds to a node $u \in \mathcal{C}_T$ when v and u have attributes that are associated with corresponding columns via the table semantics. Second, we take into consideration the following: (i) For a pair of nodes (v_1, v_2) in $\mathcal{C}_S$ and a pair of nodes (u_1, u_2) in $\mathcal{C}_T$, with v_1 corresponding to u_1 and v_2 corresponding to u_2, if there is to be a connection between v_1 and v_2 then it should be "semantically similar" or at least "compatible" to the connection between u_1 and u_2. The compatibility is decided by either the cardinality constraints of the connections imposed on the corresponding participants or the semantic type of the connections, e.g., **ISA** and **partOf**. For instance, a source relationship between v_1 and v_2 with lower bound 2 of the cardinality for v_2 is not compatible with a target relationship between u_1 and u_2 with upper bound 1 of the cardinality for u_2. The second part of Example 6.3.3 also uses **partOf** semantics to pair up compatible relationships. (ii) Since columns appearing in the same table are assumed to represent particularly relevant semantic connections between the concepts carrying the respective attributes, there is a preference that the CSGs use edges from the pre-selected s-trees. (iii) To the extent that there are choices available, we want the CSG to represent "intuitively meaningful concepts/queries". (iv) All things being equal, we want the CSG to be compact – as per Occam's principle.

In relational database, there appears to be consensus that observation (iii) favors the joins in the query to be lossless. Previous research on graphical querying of ER diagrams [ZM83] (see Example

6.4.1) indicates that *functional trees* in such diagrams correspond to lossless joins. Formally, a functional tree $\mathcal{F}$ containing a set of nodes $\{v_1, v_2, ..., v_n\}$ is a tree with a root u such that all paths from u are functional. (Such a tree is formally a Steiner tree.) The preference for functional trees is motivated by the fact that functional properties in the CM determine functional dependencies, and hence the application of the er2rel design to a functional tree gives rise to a set of relational tables whose join is lossless. Combining this with observation (iv), we are led to seek *minimal functional trees* containing, as a subset, the nodes in $\mathcal{C}_S$ ($\mathcal{C}_T$). Interestingly, Wald & Sorenson [WS84], while considering the problem of querying ER diagrams, also suggested using minimal-cost Steiner trees, but in this case passing, if necessary, through non-functional edges, whose individual cost is greater than the sum of all the functional edges (see Example 6.4.2).

Note also that meaningful queries should not be equivalent to $false$, so we will eliminate CSGs that include an ISA edge from a class node C to its parent and then an ISA$^-$ edge to a node D corresponding to a disjoint subclass from C.

We now begin to present the algorithm, which starts by finding a CSG in one side and constructs a semantically "similar" CSG in the other side. For ease of presentation, we assume that we always start from the target side, and then try to find a similar CSG in the source. There are two subcases:

- Case A: The target CSG D_2 is known, e.g., it is the s-tree associated with a single table.
- Case B: The target CSG is to be constructed itself.

Case A. We use the following example to illustrate the construction of a similar CSG in the source when the target CSG D_2 is given.

Example 6.5.2. Consider an example involving source schema with tables control(proj,dept) and manage(dept,mgr), and target table proj(pnum, dept, emp). Suppose the correspondences given are v_1:control.proj$\leftrightsquigarrow$proj.pnum, v_2:control.dept$\leftrightsquigarrow$proj.dept, and v_3:manage.mgr$\leftrightsquigarrow$proj.emp. Figure 6.9 provides the semantics of target table proj as the graph, rooted at Proj, while the semantics of the source tables are subgraphs `Project` `---controlledBy->--` `Department` and

`Department` `---hasManager->--` `Employee`

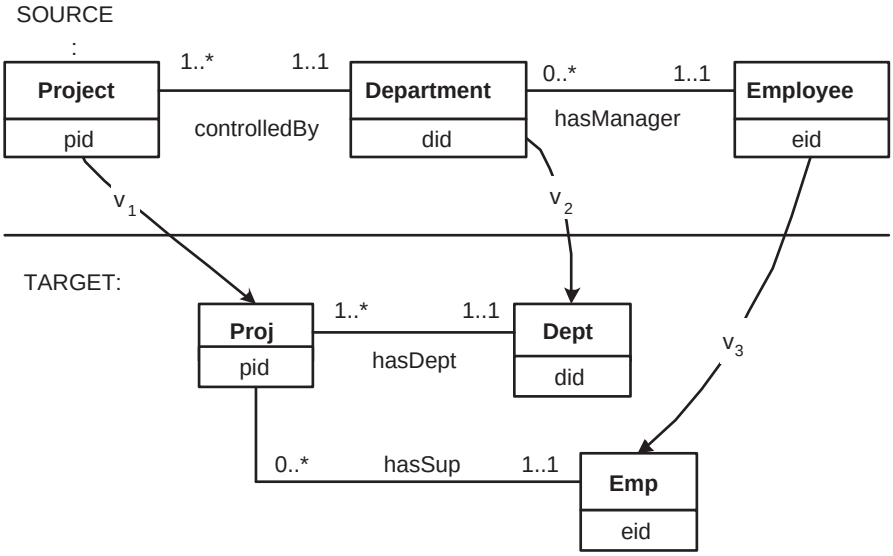

Figure 6.9: Input to Example 6.5.2

In Figure 6.9, the correspondences are lifted to correspondences between the associated class nodes.

Notice that the target CSG is an *anchored s-tree*, where the anchor is Proj, and the path from the anchor to every other node is functional. This leads us to believe that a "similar" CSG in the source should be a functional tree with a root corresponding to the anchor.

Case A.1 Suppose for the anchor Proj in the target we find a corresponding node in the source, in this case Project. Then try connect it to every other node that has a correspondence to the target CM, (Department and Employee in this case) using minimal cost functional paths. Since observation (ii) above directs us to follow edges in pre-selected trees as much as possible, the edges in pre-selected trees do not contribute to the cost of functional paths. At last, we choose the functional tree(s) satisfying the following conditions as the CSGs in the source: (1) having the minimal cost and (2) containing the most number of edges in the pre-selected s-trees. Each of such functional trees is rooted at the node corresponding to the anchor in the target. In this example, the tree `Project` `---controlledBy->--` `Department` `---hasManager->--` `Employee` is the

CSG in the source that matches the target CSG.

Case A.2 If a user only specifies correspondences v_2 and v_3 (v_1 is missing), then we can no longer find a corresponding root in the source; we are nonetheless seeking an CSG in the source that is a functional tree. In this case, we look for all functional trees in the source that contain the nodes in $\mathcal{C}_S$. Such trees should be as small as possible, hence, minimal functional trees.

In this example, we would return the same anchored tree as above. Note that even if there were another class Intern in the source graph, and a functional relationship `Intern ---works_on->-- Project`, the functional tree rooted at Intern would not be returned because it is not minimal: the functional tree rooted at Project already contains the necessary nodes.

Note that it is this technique that finds the appropriate answer for Example 6.3.2. Suppose for the source tables programmer(ssn, name, acnt) and engineer(ssn, name, site), the respective s-trees in the CM graph are `Employee --->isa-- Programmer` and `Employee --->isa-- Engineer`. These two s-trees are also pre-selected s-trees. Let the tree containing the two ISA relationships be the s-tree for the target table employee(eid, name, site, acnt). To connect the marked class nodes Programmer and Engineer using edges in the pre-selected s-trees in terms of a minimal functional tree in the CM graph, we could take Employee as the root and use the ISA edges as the functional links. The resulted tree gives rise to the desired algebraic expression over the source tables, i.e.,

$$\text{programmer(ssn, name1, acnt)} \bowtie \text{engineer(ssn, name2, site).}$$

Suppose that the nodes in C_S are not covered by a single (minimal) functional tree in the source CM graph. Then, in Case A.1, we connect as many nodes as possible using a single tree rooted at the node corresponding to the anchor and leave the rest unconnected. Consequently, the correspondences will be split among the tree and the remaining unconnected nodes. In Case A.2, we find the collection of trees covering different subsets of the nodes, and return each paired with the target CSG. ■

As the example below illustrates, a node in the target tree may correspond to multiple nodes in

the source graph.

Example 6.5.3. The target tree in Figure 6.10 consists of a single node Division. Using the correspondences

employee.name⇜⇝division.manager,

department.did⇜⇝division.did,

department.budget⇜⇝division.budget,

we identify two nodes in the source, Employee and Department, corresponding to Division of the target. To connect the two source nodes, we use key information. The concept Department has an attribute corresponding to the key did of the target concept Division. Therefore, we search for a functional tree rooted at Department, which has the attribute corresponding to did. As a result, the tree `Department ---hasManager->-- Employee` is returned.

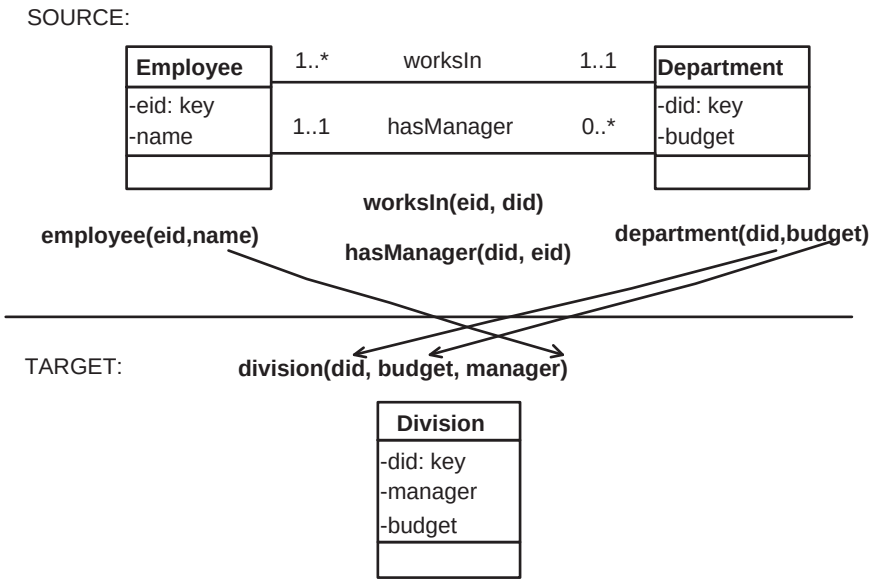

Figure 6.10: Using Key Information for Identifying the Root

However, if there is no concept in the source having an attribute corresponding to the key in the target, then we look for all minimal functional trees containing the source nodes as discussed in Example 6.5.2 Case A.2. In this case, both trees `Department ---hasManager->-- Employee` and `Employee ---worksIn->-- Department` are returned. ■

Case B. Next, we consider the case where there are several pre-selected s-trees in the target and we want to connect them. Once again, we use the idea of minimal-cost functional trees to connect the marked nodes which belong to these pre-selected trees. Consequently, we construct a set of minimal functional trees in the target. Similarly, we can construct a set of minimal functional trees in the source. From these two sets we form pairs CSGs by reverting to Case A, i.e., following heuristics such as matching the roots of tree pairs and seeking compatible connections.

We now present the algorithm for discovering CSGs in a source and a target CM graphs when given a source and a target relational schemas, their table semantics, and a set of correspondences from columns of the source schema to columns of the target schema.

Algorithm: getCSGs($\mathcal{S}$, $\mathcal{G}_S$, Σ_S, $\mathcal{T}$, $\mathcal{G}_T$, Σ_T, L)

Input: A source schema $\mathcal{S}$ associated with a CM $\mathcal{G}_S$ through the table semantics Σ_S; a target schema $\mathcal{T}$ associated with a CM $\mathcal{G}_T$ through the table semantics Σ_T; a set of correspondences L between a set of columns $L(\mathcal{S})$ in $\mathcal{S}$ and a set of columns $L(\mathcal{T})$ in $\mathcal{T}$.

Output: A set of $\langle D_1, D_2, L_M \rangle$, where D_1 and D_2 are CSGs in $\mathcal{G}_S$ and $\mathcal{G}_T$, respectively, and L_M is a set of correspondences covered by $\langle D_1, D_2 \rangle$.

Steps:

1. Let $\mathcal{C}_S$ and $\mathcal{C}_T$ be the sets of concept nodes identified by the correspondences L along with the table semantics Σ_S and Σ_T in the CM graphs $\mathcal{G}_S$ and $\mathcal{G}_T$, respectively.
   (a) Compute $\mathcal{C}_S$=onc($L(\mathcal{S})$)[1] and $\mathcal{C}_T$=onc($L(\mathcal{T})$).
2. Let $\mathcal{P}_S$ and $\mathcal{P}_T$ be the sets of pre-selected s-trees in $\mathcal{G}_S$ and $\mathcal{G}_T$, respectively.
   (a) Compute $\mathcal{P}_S$ = pstree($L(\mathcal{S})$)[2] and $\mathcal{P}_T$ = pstree($L(\mathcal{T})$).
3. Let Ans be the results set. Initialize Ans $= \emptyset$.
4. Let B be a set of CSGs in the target graph $\mathcal{G}_T$. Initialize $B = \emptyset$

[1] onc(X) is the function which gets the set of concept nodes having attributes corresponding to the columns X.
[2] pstree(L) is the function which gets the set of s-trees associated with tables with columns participating in L.

(a) If $|\mathcal{P}_T| = 1$, let $B = B \cup \mathcal{P}_T$. */*There is only one pre-selected s-tree in the target. This tree covers all nodes in $\mathcal{C}_T$.*/*

(b) Else let FT be the set of functional trees covering nodes in $\mathcal{C}_T$, initialize FT=$\emptyset$; for each node $v \in \mathcal{G}_T$

i. for each node $i \in \mathcal{C}_T$, let SP$_i$ ={p | p is a shortest functional path from v to i}. */*In computing a shortest path, the edges of the pre-selected s-trees in $\mathcal{P}_T$ do not contribute the weight of a path.*/*

ii. FT=FT $\cup$ combine(SP$_i$)[3] for all $i \in \mathcal{C}_T$.

iii. Among each set of functional tree in FT such that these trees cover the same set of nodes in $\mathcal{C}_T$, choose the tree t that has the minimum weight and contains the most number of edges in the pre-selected s-trees in $\mathcal{P}_T$, let $B = B \cup t$.

5. For each CSG $g_T \in B$, let D be a set of CSGs in the source graph $\mathcal{G}_S$. Initialize $D = \emptyset$

(a) Let u be the anchor of g_T. */*If g_T is a single pre-selected tree according to step 4.a, get the anchor of g_T by the table semantics; else the anchor of g_T is the node v specified in step 4.b.*/*

(b) Let R be the set of nodes in $\mathcal{G}_S$ corresponding to the nodes in g_T, get R through L.

(c) Let r be a node in R corresponding to u. */*If there are multiple nodes in R corresponding to u, distinguish them using* key *information of nodes in g_T, if possible.*/*

(d) If r exists

i. For each node $j \in R \backslash r$, let SP$_j$={p | a shortest functional path from r to j such that p is compatible with the path between the nodes corresponding to r and j in $\mathcal{G}_T$.} */*Using semantics like* ISA *and* **partOf** *to check the compatibility. A case considered here is that* ISA *and* **partOf** *paths are compatible with* ISA *and* **partOf** *paths, respectively.*/*

ii. $D = D \cup$ combine(SP$_j$) for all $j \in R \backslash r$.

iii. For each CSG $g_S \in D$

A. Let UC$_S$ be the set of nodes in R which are not covered by g_S. Let UC$_T$ be the set of nodes in g_T corresponding to the nodes in UC$_S$.

[3] combine is the function that merges functional shortest paths into a functional tree.

B. Let Ans=Ans $\cup\langle g_S \backslash \text{UC}_S,\ g_T \backslash \text{UC}_T,\ L_M\ \rangle$. */*$g_S \backslash UC_S$ is the graph after removing nodes in UC_S and their incident edges from g_S; the same for $g_T \backslash UC_T$.*/*

C. Let l be the correspondence linking a node v_i in UC_S and a node v_j in UC_T, let Ans = Ans $\cup\langle v_i, v_j, l\rangle$.

(e) Else */*For the functional tree g_T, we cannot find a node in the source corresponding to the anchor of g_T, so find all minimum functional trees covering R.*/*

i. Compute a set D of CSGs covering nodes in R using procedure similar to step 4.b.

ii. For each CSG $g_S \in D$, let L_M be the set of correspondences covered by $\langle g_S, g_T\rangle$; remove those leaves from g_S and g_T such that they do not participate L_M; check the compatibility of paths in g_S and g_T as in step 5.d.i.

iii. Let Ans = Ans $\cup\langle g_S, g_T, L_M\rangle$.

6. Repeat step 4 and step 5 by exchanging the source and the target in those steps but skip the CSGs that have been matched in steps 4-5.

7. If the single pre-selected s-tree identified in step 4.a is not a functional tree, then find the set of corresponding nodes in other graph and compute a minimum Steiner tree spanning the set of corresponding nodes as a matched CSG.

8. Return Ans as the result.

6.5.3 Reified Relationships

In order to represent n-ary relationships ($n > 2$) in a CM like UML, one reifies them, introducing a special class connected to the participants using so-called "roles". For example, to represent that stores sell products to persons, we introduce class Sell, with functional properties/roles seller, buyer, sold pointing to classes Store, Person and Product respectively. (See Figure 6.11.) Such reified relationship nodes will be indicated in our text by tagging their name with $\diamondsuit$, although formally this can be encoded in the CM by making such classes be subclasses of a special top-level class ReifiedRelationship. Note that classes for reified relationships may also be used to attach

descriptive attributes for relationships (e.g., dateOfPurchase). In fact, we need to use this modeling approach for binary relationships that have attributes. For ease of algorithm design, we have also chosen to represent many-to-many binary relationships, such as "person likes food", in reified form.

In terms of the formulas for table semantics, reified relationships are used in the standard way. For example, if we had table sells(sid,prodid,pid, date) whose semantics is represented by Figure 6.11, then the formula is specified as follows:

$$\begin{aligned}\mathcal{T}\text{:sells}(sid, prodid, pid, date) \quad \rightarrow \quad & \mathcal{O}\text{:Store}(x),\ \mathcal{O}\text{:Product}(y),\\ & \mathcal{O}\text{:Person}(z),\ \mathcal{O}\text{:Sell}(s),\\ & \mathcal{O}\text{:seller}(s, x),\ \mathcal{O}\text{:buyer}(s, z),\\ & \mathcal{O}\text{:sold}(s, y),\ \mathcal{O}\text{:sid}(x, sid),\\ & \mathcal{O}\text{:prodid}(y, prodid),\ \mathcal{O}\text{:pid}(z, pid),\\ & \mathcal{O}\text{:dateOfPurchase}(s, date).\end{aligned}$$

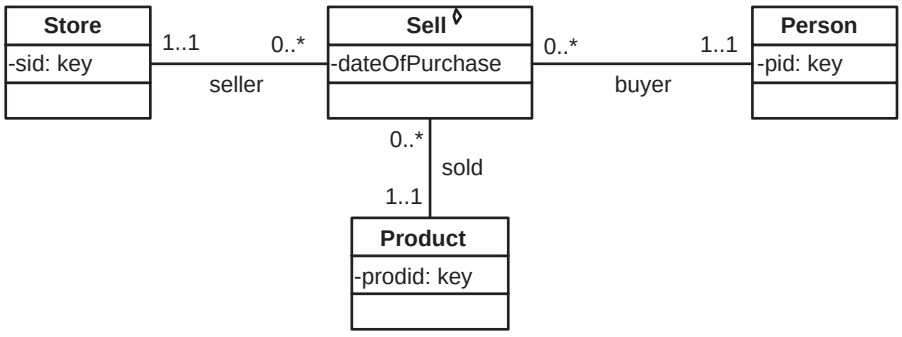

Figure 6.11: Reified Relationship Diagram

Note that cardinality constraints $0/1..1$ on inverse roles can be used to indicate those cases where an object can participate at most once in a relationship. Thus functional paths, such as `Project ---has_manager-->- Employee` can still be recognized in reified form as `Project -<-- what⁻-->- Management◇ ---who-->- Employee`.

When a reified relationship node appears in a CM graph, we make several adjustments in the mapping algorithm. First, a path of length two passing through a reified relationship node should

be counted as a path of length 1, because a reified relationship could have been eliminated, leaving a single edge.

Second, the semantic category of a target tree rooted at a reified relationship induces *preferences* for similarly rooted (minimal) functional trees in the source. This includes the anchor being many-to-many, many-to-one or one-to-one (distinguished by the cardinality restrictions on the role inverses, as in the Management example above), the number of roles (exact arity), or subclass relationship to top-level CM concept (e.g., partOf$^{\diamondsuit}$).

Third, note that non-functional relationships between entities in a CM can also be derived as the composition of edges on non-functional paths. For example, traversing the path Person ---shopsAt--- Store ---location-->- City yields a many-to-many relationship between persons and cities where the stores are located. Thus, in seeking matches in the source for a target (reified) many-to-many binary relationships between A and B, one must also consider the possibility that they appear as paths from A' to B' in the source that are not functional in either direction, where A' and B' are nodes in the source corresponding to A and B in the target, respectively. Note that using a single reified relationship as an anchor and extending this graph by functional paths from the roles, corresponds to lossless joins with the table representing the root; therefore such CSGs are preferred. More generally, we look for CSGs that minimize the number of lossy joins, by minimizing the number of *direction reversal changes* along each path. In terms of the algorithm getCSGs($\mathcal{S}$, $\mathcal{G}_S$, Σ_S, $\mathcal{T}$, $\mathcal{G}_T$, Σ_T, L), we add the following adjustment.

- If the anchor of a CSG to-be-discovered is a reified many-to-many relationship that has been identified, then we use shortest paths with minimum direction reversal changes along each path to connect the anchor to other nodes.

Example 6.5.4. [Example 6.3.1 revisited] The solution to the problem in Example 6.3.1 is then obtained as follows. The target s-tree in Figure 6.4 is a many-to-many relationship, which our algorithm represents as a reified relationship with anchor hasBookSoldAt$^{\diamondsuit}$. To find a matching CSG connecting the node Person in the source (corresponding to Author in the target) and the node Bookstore in the source (corresponding to Bookstore in the target), we look for paths connecting

them that are not functional in either direction. Note that going from one role filler to another of a reified many-many binary relationship produces exactly such a path. In this case, no such single reified node can be found in the source. So we look for longer paths, obtaining the path from Person to Bookstore through writes$^{\diamond}$, Book, and soldAt$^{\diamond}$.

■

6.5.4 Obtaining Relational Expressions

The final mapping expression requires a pair of algebraic expressions using the tables in the input relational schemas only. Therefore, we need to translate the discovered CSGs in the CM graphs into algebraic expressions over the database schemas. Consider a CM as a collection of primitive relations/predicates for its concepts, attributes and properties. The semantics of a relational table associated with the CM is a LAV expression over these predicates. Translating a discovered subgraph in the CM graph into expressions over tables associated with the CM graph becomes a query rewriting problem.

The first step of the translation is to express the CSG as a query using CM predicates. The encoding algorithm proposed in Figure 4.7 of Section 4.3 can be used for this purpose. The following example illustrates this.

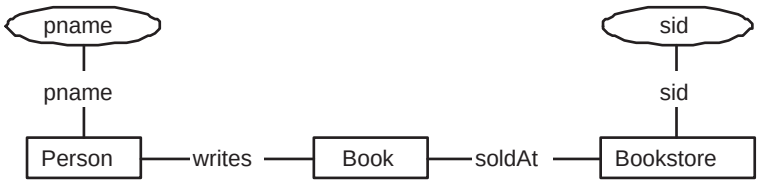

Figure 6.12: A Discovered Tree over a CM Graph

Example 6.5.5. Figure 6.12 is a fully specified CSG in the source CM of Example 6.3.1, with attribute nodes shown. (For simplicity of presentation, we revert to unreified binary relationships.) Taking Person as the root of the tree, the encoding algorithm recursively constructs a logic formula using unary predicates for the class nodes and binary predicates for the edges. An attribute node is

encoded as a fresh variable in the formula and appears in the answer tuple. Assigning a name ans to the query, we obtain

$$\begin{aligned} \mathsf{ans}(v_1,v_2) \quad \text{:-} \quad & \mathcal{O}\text{:}\mathsf{Person}(x_1),\ \mathcal{O}\text{:}\mathsf{pname}(x_1,\ v_1), \\ & \mathcal{O}\text{:}\mathsf{writes}(x_1,\ x_2),\ \mathcal{O}\text{:}\mathsf{Book}(x_2), \\ & \mathcal{O}\text{:}\mathsf{soldAt}(x_2,\ x_3),\ \mathcal{O}\text{:}\mathsf{Bookstore}(x_3), \\ & \mathcal{O}\text{:}\mathsf{sid}(x_3,\ v_2). \end{aligned}$$

■

Given the table semantics in terms of logical formulas, we rewrite the query q above to a new query q' which only mentions the tables in the relational schema by taking advantage of the object identifier information in the table semantics. The new query q' is maximally-contained (see [Hal01]) in q and should mention tables that have columns linked by the correspondences.

Example 6.5.6. For the sake of completeness, we now briefly describe a process of rewriting. We have proposed in Section 4.5 an ad-hoc approach to deriving inverse rules for each predicate in a CM, in terms of the tables in a relational schema. An essential problem resolved there is that unique internal object identifiers (e.g., arguments like x for $\mathsf{Person}(x)$) used in the CM, are not directly available in the relational tables. Formally, these are converted to Skolem functions, giving rise to formulas such as

$$\mathcal{O}\text{:}\mathsf{Person}(f(pname, age)) \text{ :- } \mathcal{T}\text{:}\mathsf{person}(pname, age).$$

when inverting a semantic specification such as

$$\mathcal{T}\text{:}\mathsf{person}(pname, age) \rightarrow \mathcal{O}\ \text{:}\mathsf{Person}(x),\ \mathcal{O}\ \text{:}\mathsf{hasName}(x, pname),\ \mathcal{O}\text{:}\mathsf{hasAge}(x, age).$$

The problem is that different tables give rise to different Skolem functions, which cannot then be joined. For this purpose, we use the key information about table semantics (see Table 4.1 of Section 4.3) in order to "unify" the various Skolem functions. So if we knew that hasName is the key of entity Person, and the body formula Φ contains $\mathsf{Person}(x) \wedge\ \mathsf{hasName}(x, z)$ then we can in fact use z instead of x as the internal identifier, and treat hasName as the identity relation.

As a result, we can rewrite the query q in Example 6.5.5 to queries that mention tables only. In our case, these include the following:

q_1': ans(v_1,v_2) :- $\mathcal{T}$:writes(v_1, y), $\mathcal{T}$:soldAT(y, v_2).

q_2': ans(v_1,v_2) :- $\mathcal{T}$:Person(v_1), $\mathcal{T}$:writes(v_1, y),
$\mathcal{T}$:Book(y), $\mathcal{T}$:soldAT(y, v_2),
$\mathcal{T}$:Bookstore(v_2).

q_3': ans(v_1,v_2) :- $\mathcal{T}$:Person(v_1) $\mathcal{T}$:writes(v_1, y),
$\mathcal{T}$:soldAT(y, v_2), $\mathcal{T}$:Bookstore(v_2).

Since q_1' does not mention tables person(pname) and bookstore(sid) that are linked by the correspondences, and q_2' is contained in q_3', q_1' and q_2' are eliminated. The body of the query q_3', converted to relational algebra in the standard way, is returned as the algebraic expression.

■

6.6 Experimental Evaluation

We now report on experimental results that evaluate the performance of the proposed approach. We show that this approach works reasonably in a number of cases, and achieves better results than the RIC-based techniques for discovering a complex mapping expression among marked elements in the schemas. The implementation is in Java and all experiments were performed on a PC-compatible machine with a Pentium IV 2.4GH CPU and 512MB memory.

Datasets: We considered a variety of domains. For each, a pair of relational schemas developed independently was used for testing. We ensured that the CMs associated with the pair of schemas were also mutually independent, by using different domain CMs or the different ER conceptual models used for deriving the independent schemas. We describe them briefly below. All the schemas and CMs used in our experiments are available at [An06].

The first three pairs of schemas were obtained from Clio's test datasets [PVM$^+$02]. DBLP 1&2 are the relational schemas for the DBLP bibliography. They are associated with the Bibliographic

Schema	#tables	associated CM	#nodes in CM	#mappings tested	time (sec)
DBLP1	22	Bibliographic	75	6	0.072
DBLP2	9	DBLP2 ER	7		
Mondial1	28	factbook	52	5	0.424
Mondial2	26	mondial2 ER	26		
Amalgam1	15	amalgam1 ER	8	7	0.14
Amalgam2	27	amalgam2 ER	26		
3Sdb1	9	3Sdb1 ER	9	3	0.105
3Sdb2	9	3Sdb2 ER	11		
UTCS	8	KA onto.	105	2	0.384
UTDB	13	CS dept. onto.	62		
HotelA	6	hotelA onto.	7	5	0.158
HotelB	5	hotelB onto.	7		
NetworkA	18	networkA onto.	28	6	0.106
NetworkB	19	networkB onto.	27		

Table 6.1: Characteristics of Test Data

ontology and an ER model reverse engineered from the DBLP2 schema, respectively. Mondial 1&2 are databases about countries and their various features, where Mondial1 is associated with the CIA factbook ontology and Mondial2 is reverse engineered. Amalgam 1&2 are test schemas developed by students and used in the Clio evaluations. They associate with different conceptual models. 3Sdb 1&2 are two versions of a repository of data on biological samples explored during gene expression analysis [JTBM06]. UTCS and UTDB are databases for the CS department and the DB group at the University of Toronto. They were used in our previous study of semantics discovery, so their semantics are available now. Finally, we chose two pairs of ontologies from the I3CON conference[4]. These ontologies were used for the ontology alignment competition and demonstrate a certain degree of modeling heterogeneity. We forward engineered them into relational schemas for testing our techniques. As shown in Table 6.1, the test data have a variety of complexities.

Methodology: We compared the semantic approach, presented in this chapter, with the RIC-based technique illustrated in Example 6.3.1, which, recall, creates logical relations by chasing RICs constraints, and derives mappings from pairs of source-target logical relations covering some

[4]http://www.atl.external.lmco.com/projects/ontology/i3con.html

correspondences. Since in its raw form, the chase generates maximal sets of columns that can be grouped by lossless joins, we first applied a heuristic that removed any unnecessary joins — ones that did not introduce new attributes not covered by correspondences. (This is one optimization recently described in [FHH$^+$06].)

The comparison tries to focus on the intrinsic abilities of the methods. Each experiment consists of a manually created non-trivial "benchmark" mapping between some pair of schemas. (A trivial mapping is from a single source table to a single target table.), together with correspondences involving column names in it. These manually-created mappings are used as a "gold standard" to compare the mapping performance of the different methods.

Measures: We use $precision$ and $recall$ to measure the performance of the methods. For a given schema pair, let P be the set of mappings generated by a method for a given set of correspondences. Let R be the set of manually-created mappings for the same given set of correspondences. The two measures are computed as: $precision = \frac{|P \cap R|}{|P|}$ and $recall = \frac{|P \cap R|}{|R|}$. We compute the average precision and average recall over all tested mapping cases.

We believe it is instructive to give more details about how we calculate these measures. For each test case, R contains the manually-created non-trivial benchmark mapping expression consisting of a connection in the source and a connection in the target. In evaluating the generated mappings for each method, we seek the same pair of connections, considering others that do not match the benchmark completely as "incorrect" mappings. For instance, in Example 6.3.1, even if there were target tables for author2, store2, and the RIC-based techniques recovered mappings $\langle$person,author2$\rangle$ and $\langle$store,store2$\rangle$, recall and precision would have been 0 because no non-trivial mappings were found. (Note that the semantic method can also find trivial mappings.)

Results: First, the times used by the semantic approach for generating the mappings (in algebraic expressions) in the tested schemas are insignificant. The last column of Table 6.1 shows that it took less than one second. This is comparable with the RIC-based technique, which also took less than one second for mapping generation in our experiments. Next, in terms of the measures, Figure 6.13 compares the average precisions of semantic and the RIC-based technique for all the domains.

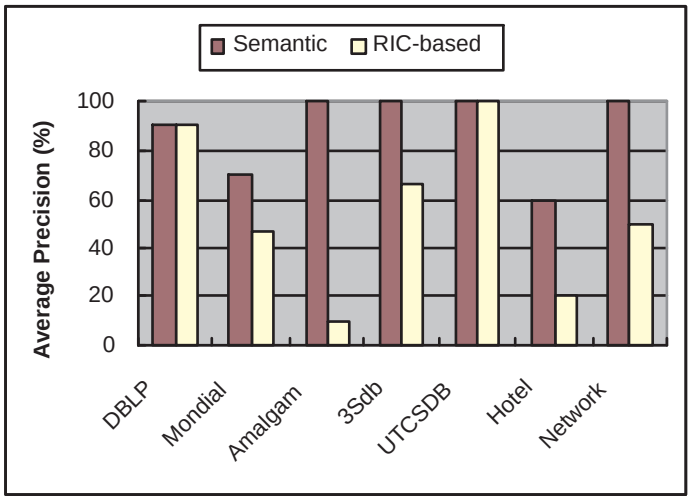

Figure 6.13: Average Precision

Figure 6.14 compares the average recalls.

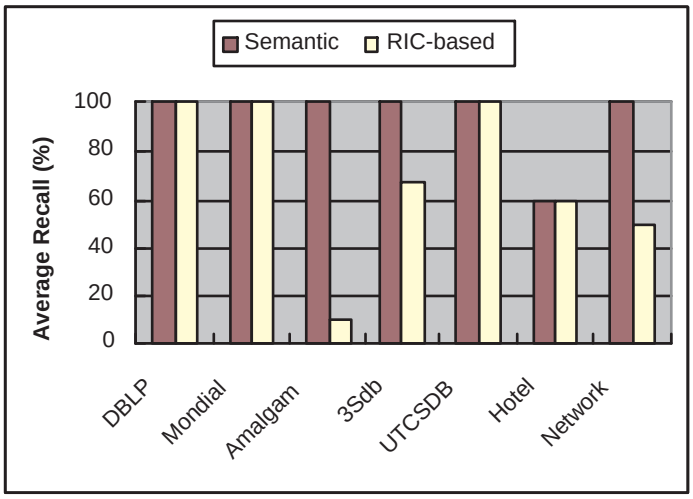

Figure 6.14: Average Recall

The results show that in general, the semantic approach performed at least as well as the RIC-based technique for the test datasets. The measures of recall show that the semantic approach did not miss any correct mappings that were predicted by the RIC-based technique (since it got *all* the mappings sought), and made significant improvements in some cases. Moreover, Figure 6.13 shows

that the semantic approach had significantly improved precision in some cases.

Discussion: Many of the experimental schemas we have found do not have complicated semantics, and therefore would not provide differing results unless somewhat complex correspondences and mappings were sought. Most of the differences in this particular experimental set were due to situations such as the one illustrated in Example 6.3.2.

6.7 Discussion

We have proposed here an approach to discovering mapping expressions between a pair of relational schemas. The approach starts from simple table column correspondences and utilizes the semantics of the tables, expressed through connections to conceptual models. We first showed several cases where the current solutions for discovering mappings (based on referential integrity and key constraints) does not produce the best results. We then developed an algorithm for discovering plausible mappings at the conceptual level, and translated them into sketches of relational level mappings. Intuitively, this algorithm replaces the use of RICs by the notion of ”minimal functional tree” in the CM, which, interestingly, appear to be related through the theory of Universal Relations and lossless joins. Experimental results demonstrated that the semantic approach achieved a generally better performance in recovering complex mapping expressions on test datasets drawn from a variety of domains.

Compared to the traditional schema mapping techniques, e.g., Clio, our solution has an obvious limitation, that is, it assumes the semantics of the schemas being mapped are available. Given the additional significant work that has gone into the Clio tool, it may be best to view the present work as being complementary and embedded: if the semantics of the schemas is available or can be reconstructed with low cost using the MAPONTO tool, then the present technique could be used inside Clio to provide some better candidate mappings. The exact details of such a merger remain to be worked out.

In terms of future work, we are planning to work on the representation of all conjunctive queries over the CM, as well as safe negation in s-trees/s-graphs. As shown in Chapter 4, a more careful

look at the tree provides hints about when joins should really be treated as outer-joins (e.g., when the minimum cardinality of an edge being traversed is 0, not 1); such information could be quite useful in computing more accurate mappings, expressed as nested tuple-generating dependencies. Investigating the related problem of finding complex semantic mappings between two CMs, given a set of element correspondences, is another future research direction.

6.8 Summary

Based on our study on discovering semantics for database schemas in previous two chapters, we developed a new approach to schema mapping by using the semantics of schemas in this chapter. The essential point is to find a pair of associations in the source and the target schemas such that the pair are “semantically similar”. For doing this, we explored various constructs in the CMs associated with schemas. These constructs include cardinality constraints, types of relationships, and graph structures. We believe that what we have done in this chapter will lead to more work in exploring and utilizing semantics of data.

Chapter 7

Conclusions and Future Directions

Discovering semantic mappings between different data representations is a critical step for many information management applications. Since manual creation of semantic mappings is labor-intensive and error-prone, it is important to develop tools to automate the process. This dissertation has contributed to both developing tools for semantic mapping creation and understanding semantics for database schemas. In this chapter, we recapitulate the main contributions of the dissertation and discuss directions for future research.

7.1 Main Contributions

This dissertation makes two major contributions: (i) discovering semantics for widely used database schemas in terms of CMs and (ii) using the semantics for discovering mapping expressions between database schemas.

We approach the problem of discovering semantics for database schemas by aligning the modeling constructs in schemas and given CMs. The alignment is guided by well-known database design principles. The underlying assumption is that a mapping expression relates two "semantically similar" associations in different models.

To increase the chance of getting accurate mapping results, we request from the user simple

element correspondences as an additional input for the mapping algorithm. This approach is supported by other mapping tools, and simple element correspondences can be specified manually or by schema matching tools. Starting from the correspondences between the elements in a schema and a CM, we directed our attention to find an association among elements in the schema and an "semantically similar" association among elements in the CM. First, we meticulously analyzed the standard database design methodology. We uncovered a number of notions related to the transformation process in addition to a set of standard rules for generating schema structures from a conceptual model. These notions, e.g., anchor, play a central role in the development of the semantic mapping discovery algorithm.

Second, we constructed a CM graph from a CM by taking cardinality constraints and relationship types into consideration. We differentiate functional paths from non-functional paths for the purpose of interpreting the key and foreign key constraints in database schema. Third, focusing on the schemas that could have been generated by the standard design methodology from conceptual models, we develop an algorithm that is "complete" and "sound". This result gives us a theoretical confidence for judging the performance of the algorithm on some "regular" database schemas. To evaluate the algorithm on a wide variety of practical schemas, we finally implemented and tested the algorithm in our prototype tool MAPONTO. One lesson we have learned from the experimental study is that the tool is useful because not only it can produce correct answers for many cases, but also it could reduce the burden that would be endured by human even though the answers are incorrect. The user still could benefit from the incorrect answers by debugging them into the desired ones instead of specifying the correct answers completely from scratch.

Our second major contribution is made by taking advantage of the semantics of database schemas that would have been discovered by the MAPONTO tool for improving schema mapping. We are motivated by the observation that traditional schema mapping approaches relying on schema constraints and structures sometimes do not produce desired mapping relationships between a given source and a given target schemas. It is in part because logical schemas are "semantically impoverished"; therefore, it is inherently difficult for computerized tools to discover mappings between heterogeneous schemas. Moreover, traditional mapping techniques would first find logical associ-

ations in the source and target schemas and then would pair each logical association in the source with each logical association in the target. As a result, the number of candidate mappings would be huge since every combination of a logical association in the source and a logical association in the target is considered as a candidate. The user may be overwhelmed by the results generated by the traditional mapping tools.

In order to improve schema mapping, we believe that multiple information sources about schemas have to be exploited. As we have studied the issues of semantics for database schemas, we exploit the information about schema semantics expressed in semantic mappings to CMs. Although we have developed a tool for discovering semantics for schemas in terms of given CMs, many database schemas were developed independently from different conceptual models. We therefore do not assume that schemas are connected at the CM level. We focus on exploiting the semantic information encoded in different CMs for discovering possible relationships between schemas. The underlying assumption is the same as that for discovering schema semantics, that is, seeking for a pair of "semantically similar" associations in different schemas. To do this, the CMs are treated as graphs and the semantics of each relational table is a tree called s-tree. Given a set of correspondences between schema elements, we first identify sets of concept nodes in the CM graphs. The algorithm then focuses on finding compatible conceptual subgraphs in both CM graphs. Finally, these pairs of compatible subgraphs are translated into algebraic expressions over the schema elements. A pair of algebraic expression gives rise to a schema mapping expression. Our experimental results show that the semantic approach outperforms the traditional techniques in terms of both recall and precision.

The key innovations that we have made in developing the solution to semantic schema mapping is that we bring the necessity of explicit representation of semantics for schemas to the forefront of data integration and we exploit a kind of semantics for improving schema mapping. We expect more work ahead for representing and exploiting the full semantics of various database schemas.

7.2 Future Directions

Although we have made significant progress on the problem of discovering and using semantics for database schemas, we believe that what we have tackled is just a tip of the iceberg representing the long-standing problem of understanding data semantics. To achieve the goal of seamless semantic integration over heterogeneous and multiple information systems, substantial work remain. We now discuss several immediate future research directions.

Mapping between CMs One problem is to discover complex semantic mappings between CMs. Although we have developed solutions for discovering a semantic mapping from database schemas to CMs and for discovering schema mapping, it is unclear whether the solutions are directly applicable for discovering semantic mappings between CMs. What is clear is that CMs tend to have more complex structure and richer semantics. A CM language often provides a rich set of constructs for modeling a subject matter. This increases the difficulty in discovering semantic mappings between CMs because there are more variations for modeling the same real world object. Nonetheless, many applications demand establishing semantic mappings between various CMs. A prominent example is the Semantic Web, where data are annotated with domain ontologies. It is impossible that a single ontology or a few monolithic ontologies would dominate the Semantic Web. Instead, data would be annotated with numerous, different ontologies. A pressing problem in the realization of the Semantic Web is to connect these ontologies by semantic mappings.

Although ontology mapping has attracted a steady attention since the advent of the Semantic Web, little effort has been put into deriving complex mapping expressions between ontologies. Most of the current solutions focus on producing correspondences between single elements in ontologies, e.g., a concept C_1 in an ontology is a synonym of the concept C_2 in another ontology. A semi-automatic and interactive approach similar to what we have developed in this dissertation would be appropriate for deriving complex mapping expressions for CMs.

Semantic Mapping Management The second direction is mapping management including maintaining semantic mappings associated with design process and adapting mappings when domain models evolve. As we have observed, current practices in database design do not keep the con-

ceptual models for late use. One way to reduce legacy data is to maintain the semantic mapping associated with database design process. A key problem of the maintenance is that schemas change constantly due to the open and dynamic application environment. When a schema evolves, the corresponding CM and the semantic mapping also need to adapt the schema's evolution in order to maintain the validity of the mapping.

Semantic Mapping Composition Third, composing a series of semantic mappings to reach a direct mapping is a natural application. Suppose we have a situation where data providers willingly connect their data sources to acquainted sources and these sources are connected to other acquaintances. A similar situation is that different databases are derived from a common CM or that database schemas are connected at their CM level. In almost all these situations, semantic mapping composition plays an important part in direct information exchange between sources without direct mappings. Current solutions for mapping composition focus on mappings between relational database schemas. Such a mapping often is specified in the form of source-to-target tuple generating dependency (see Section 2.1). It is worth investigating the composition of semantic mappings from two different database schemas to a CM with rich semantics. Moreover, composing a semantic mapping from a database schema to a CM with a semantic mapping between two CMs will generate a new semantic mapping for the database schema.

Integrating Schema Matching Tools The fourth future work direction is to integrate the schema matching tools with our tool to achieve fuller automation. Simple correspondences between elements in different domain models play a key role in our solutions to both the problem of discovering semantics and the problem of using semantics for database schemas. So far we have assumed that correspondences are available and there are no questions about the accuracy of correspondences. Almost all schema matching tools, however, are semi-automatic, needing human intervention in choosing desired results. Integrating such schema matching tools with our tools faces many challenges. The biggest one is that the correspondences generated by a schema matching tool may be ambiguous or incorrect. If we expect to achieve full automation, these correspondences would be fed into our tool as part of the input. As a result, the final mapping expressions may be meaningless and useless. We believe that the semantic information available in both schemas and CMs can be ex-

ploited to inspect the input correspondences in order to generate meaningful mapping expressions. Substantial work remains.

Exploiting Data Instances for Ordering the Candidates A fifth direction is to prioritize the mapping candidates according to data instances. A database often has data instances accompanying its schema. A mapping relating a domain model M_1 to another domain model M_2 can be applied to an instance of M_1 as a query, generating a new instance for M_2. If there is an existing instance of M_2, we could compare the new generated instance with the existing one in terms of some "distance" criteria between them. The closer the two instances are, the higher the priority is assigned to the mapping. In addition, comprehensive machine learning techniques can be used to discover semantic patterns for the same purpose.

Better User Interface for Interaction Finally, we need better user interfaces for efficient interaction. The solutions we have developed must interact with the user in order to arrive at the final correct mappings. We consider the development of user-friendly interfaces for efficient user interaction is one of the most important problems in developing various mapping tools. Our own experience working with the MAPONTO tool shows that the user can easily get frustrated in browsing large scale and complex schemas and CMs for verifying the mappings generated by the tool. Current data integration systems may involve hundreds and thousands different databases and conceptual models. Consequently, a great number of mappings need to be verified manually before meaningful data integration is provided. A fundamental criterion in developing interfaces for efficient user interaction is to minimize the user input and maximize the yield of the input.

Bibliography

[ABFS02] B. Amann, C. Beeri, I. Fundulaki, and M Scholl. Ontology-based integration of XML web resources. In *Proceedings of the International Conference on Semantic Web (ISWC)*, pages 117–131, 2002.

[ABM05a] Y. An, A. Borgida, and J. Mylopoulos. Constructing Complex Semantic Mappings between XML Data and Ontologies. In *Proceedings of the International Conference on Semantic Web (ISWC)*, pages 6–20, 2005.

[ABM05b] Y. An, A. Borgida, and J. Mylopoulos. Inferring Complex Semantic Mappings between Relational Tables and Ontologies from Simple Correspondences. In *Proceedings of International Conference on Ontologies, Databases, and Applications of Semantics (ODBASE)*, pages 1152–1169, 2005.

[ABM06] Y. An, A. Borgida, and J. Mylopoulos. Discovering the Semantics of Relational Tables through Mappings. *Journal on Data Semantics*, VII:1–32, 2006.

[ABMM07] Y. An, A. Borgida, R. J. Miller, and J. Mylopoulos. A Semantic Approach to Discovering Schema Mapping Expressions. In *Proceedings of International Conference on Data Engineering (ICDE)*, 2007.

[AD98] S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 254–263, 1998.

[AKS96] Y. Arens, C. A. Knoblock, and W. Shen. Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems*, 6(2-3):99–130, 1996.

[AL05] M. Arenas and L. Libkin. XML Data Exchange: Consistency and Query Answering. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 13–24, 2005.

[ALM02] F. A., C. Li, and P. Mitra. Answering queries using views with arithmetic comparisons. In *ACM Symposium on Principles of Database Systems*, pages 209–220, 2002.

[AMB06] Y. An, J. Mylopoulos, and A. Borgida. Building Semantic Mappings from Databases to Ontologies. In *Proceedings of American Association for Artificial Intelligence (AAAI)*, 2006.

[An06] Y. An. http://www.cs.toronto.edu/˜yuana/research/maponto/ schemaMapping. 2006.

[And94] M. Andersson. Extracting an entity relationship schema from a relational database through reverse engineering. In *Proceedings of International Conference on Conceptual Modeling (ER)*, 1994.

[B+02] F. Baader et al. *The Description Logic Handbook*. Cambridge University Press, 2002.

[BBB+97] R. J. Bayardo, W. Bohrer, R. Brice, A. Cichocki, G. Fowler, A. Helai, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. InfoSleuth: Agent-based semantic integration of information in open and dynamic environments. In *Proceedings of ACM SIGMOD*, pages 195–206, 1997.

[BCH+05] A. Bonifati, E. Q. Chang, T. Ho, V. S. Lakshmanan, and R. Pottinger. HePToX: Marring XML and Heterogeneity in Your P2P Databases. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*, pages 1267–1270, 2005.

[BE97] R. Barquin and H. Edelstein. *Planning and Designing the Data Warehouse*. Prentice-Hall, 1997.

[Ber03] P. Bernstein. Applying Model Management to Classical Meta Data Problems. In *CIDR*, 2003.

[BGK+02] P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Database Management for Peer-to-Peer Computing: A Vision. In *The International Workshop on the Web and Databases*, 2002.

[BHP00] P. A. Bernstein, A. Y. Halevy, and R. A. Pottinger. A vision for management of complex models. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 29(4):55–63, 2000.

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.

[BLN86] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of Methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–264, 1986.

[BM04] A. Borgida and J. Mylopoulos. Data Semantics Revisited. In *Proceedings of the Workshop on Semantic Web and Databases (SWDB) in Conjunction with the International Conference on Very Large Data Bases (VLDB)*, pages 9–26, August 2004.

[BMS84] M. L. Brodie, J. Mylopoulos, and J. W. Schmidt. *On Conceptual Modeling*. Springer-Verlag, New York, 1984.

[BR00] P. A. Bernstein and E. Rahm. Data warehouse scenarios for model management. In *International Conference on Conceptual Modeling / the Entity Relationship Approach*, pages 1–15, 2000.

[BS85] R. J. Brachman and J. Schmolze. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9(2), 1985.

[BSZ03] P. Bouquet, L. Serafini, and S. Zanobini. Semantic Coordination. In *Proceedings of the International Conference on Semantic Web (ISWC)*, 2003.

[Cas83] M. A. Casanova. Designing Entity-Relationship Schemas for Conventional Information Systems. In *Proceedings of International Conference on Entity-Relationship Approach (ER)*, 1983.

[CBS94] R. H. L. Chiang, T. M. Barron, and V. C. Storey. Reverse engineering of relational databases: extraction of an EER model from a relational database. *Data and Knowledge Engineering*, 12:107–142, 1994.

[CCGL02] A. Cali, D. Calvanese, G. De Giacomo, and M. Lenzerini. Data integration under integrity constraints. In *Proc. of CAiSE'02, Toronto, Canada*, 2002.

[CD97] D. Chaudhuri and U. Dayal. An Overview of Data Warehoushing and OLAP Technology. In *SIGMOD Record, 26(1)*, pages 65–74, 1997.

[CE87] B. Czejdo and D. W. Embley. An approach to schema integration and query formulation in federated database systems. In *the 3rd IEEE Conference on Data Engineering*, pages 477–484, 1987.

[CGL+01a] D. Calvanese, G. D. Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Data integration in data warehouse. *Cooperative Information Systems*, 10(3):237–271, 2001.

[CGL01b] D. Calvanese, G. De Giacomo, and M. Lenzerini. Ontology of integration and integration of ontologies. In *Description Logics*, 2001.

[Che75] P. Chen. The Entity-Relationship Model: Towards a Unified View of Data. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*, 1975.

[CHS91] C. Collet, M. N. Huhns, and W.-M. Shen. Resource integration using a large knowledge base in Carnot. *IEEE Computer*, 24:55–62, Dec 1991.

[CM77] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. of the 9th ACM Symp. on Theory of Computing*, pages 77–90, 1977.

[Cod70] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communication of the ACM*, 6(13), 1970.

[CV92] S. Chaudhuri and M. Y. Vardi. On the equivalence of recursive and nonrecursive Datalog programs. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 55–66, 1992.

[DA83] S. R. Dumpala and S. K. Arora. Schema Translation Using the Entity-Relationship Approach. In *Proceedings of International Conference on Entity-Relationship Approach (ER)*, 1983.

[DDH01] A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: A machine learning approach. In *SIGMOD'01*, 2001.

[DGL00] O. M. Duschka, M. R. Genesereth, and A. Y. Levy. Recursive Query Plans for Data Integration. *Journal of Logic Programming*, 43(1):49–73, 2000.

[DHM+04] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity Search for Web Services. In *Proc. of International Conference on Very Large Data Bases (VLDB)*, 2004.

[DLD+04] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering Complex Semantic Matches between Database Schemas. In *Proceedings of the ACM SIGMOD*, pages 383–394, 2004.

[DMDH02] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *Proc. of the International WWW Conference*, 2002.

[DMQ03] D. Dou, D. McDermott, and P. Qi. Ontology Translation on the Semantic Web. In *Proceedings of Intl. Conference on Ontologies, Databases, and Applications of Semantics (ODBASE)*, 2003.

[DNH04] A. Doan, N. Noy, and A. Halevy. *Introduction to the Special Issue on Semantic Integration*. ACM SIGMOD Record 33(4), 2004.

[Doa02] A. Doan. *Learning to Map between Structured Representations of Data* . Ph.D. Thesis, University of Washington, 2002.

[DP02] M. Dahchour and A. Pirotte. The Semantics of Reifying n-ary Relationships as Classes. In *Information Systems Analysis and Specification*, pages 580–586, 2002.

[DR02] H. H. Do and E. Rahm. COMA - a system for flexible combination of schema matching approaches. In *Proceedings of the International Conference on Very Large Data bases (VLDB)*, 2002.

[EM01] D. W. Embley and W. Y. Mok. Developing XML Documents with Guaranteed "Good" Properties. In *Proceedings of International Conference on Conceptual Modeling (ER)*, 2001.

[Fag06] R. Fagin. Inverting Schema Mappings. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, 2006.

[FG92] M. M. Fonkam and W. A.G. Gray. An approach to eliciting the semantics of relational databases. In *Proceedings of CAiSE*, 1992.

[FHH+06] A. Fuxman, M. A. Hernandez, H. Ho, R. J. Miller, P. Papotti, and L. Popa. Nested Mappings: Schema Mappings Reloaded. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*, 2006.

[FHM05] M. Franklin, A. Halevy, and D. Maier. From Databases to Dataspaces: A New Abstraction for Information Management. *SIGMOD Record*, 34(4), 2005.

[FKMP03] R. Fagin, P. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. In *Proceedings of International Conference on Database Theory (ICDT)*, 2003.

[FKP03] R. Fagin, P. Kolaitis, and L. Popa. Data Exchange: Getting to the Core. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, 2003.

[FKPT04] R. Fagin, P. Kolaitis, L. Popa, and W. Tan. Composing Schema Mappings: Second-Order Dependencies to the Rescue. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, 2004.

[FLM99] M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *the National Conference on Artificial Intelligence*, 1999.

[FW04] D. C. Fallside and P. Walmsley. XML Schema Part 0: Primer Second Edition. In *W3C Recommendation*, http://www.w3.org/TR/xmlschema-0/, October 2004.

[GGM+02] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider. Sweetening Ontologies with DOLCE. In *Proceedings of 13th International Conference on Knowledge Engineering and Knowledge Management, Ontologies and the Semantic Web*, pages 166–181, 2002.

[GHS+95] M. J. Garey, L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, M. Flickner, A. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J. H. Williams, and E. L. Wimmers. Towards Heterogeneous Multimedia Information Systems: The Garlic Approach. In *Proceedings of the 5th International Workshop on Research Issues in Data Engineering - Distributed Object Management (RIDE-DOM)*, pages 124–131, 1995.

[GLR99] F. Goasdoue, V. Lattes, and M. Rousset. The Use of Carin Language and Algorithm for Information Integration: The PICSEL Project. *International Journal of Cooperative Information Systems*, 9(4):383–401, 1999.

[GMPQ+97] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *J. of Intelligent Information Systems*, 8(2):117–132, 1997.

[GR04] F. Goasdoue and M.-C. Rousset. Answering Queries using Views: a KRDB Perspective for the Semantic Web. *ACM TOIT*, 4(3):255 – 288, 2004.

[Hai98] J.-L. Hainaut. *Database reverse engineering*. http://citeseer.ist.psu.edu/article/hainaut98database.html, 1998.

[Hal00] A. Y. Halevy. Theory of answering queries using views. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 29(4):40–47, 2000.

[Hal01] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal: Very Large Data Bases*, 10(4):270–294, 2001.

[Hal05] A. Halevy. Why Your Data Won't Mix. *ACM Queue*, 3(8), 2005.

[HC06] B. He and K. C.-C. Chang. Automatic Complex Schema Matching across Web Query Interfaces: A Correlation Mining Approach. *ACM Transactions on Database Systems*, 31(1), 2006.

[HEH+98] J. Henrard, V. Englebert, J.-M. Hick, D. Roland, and J.-L. Hainaut. Program Understanding in Databases Reverse Engineering. In *Database and Expert Systems Applications (DEXA)*, pages 70–79, 1998.

[HFM06] A. Halevy, M. Franklin, and D. Maier. Principle of Dataspace Systems. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, 2006.

[HIMT03] A. Halevy, Z. G. Ives, P. Mork, and I. Tatarinov. Piazza: data management infrastructure for semantic web application. In *Proceedings of International Conference on World Wide Web (WWW)*, 2003.

[HIST03] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema Mediation in Peer Data Management Systems. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2003.

[HRW92] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem*. Annals of Discrete Mathematics, 53, 1992.

[Hul84] R. Hull. Relative information capacity. In *3rd ACM SIGACT-SIGMOD*, pages 97–109, 1984.

[Joh94] P. Johannesson. A method for transforming relational schemas into conceptual schemas. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 190–201, 1994.

[JS96] T. H. Jones and I.-Y. Song. Analysis of Binary/ternary Cardinality Combinations in Entity-Relationship Modeling. *Data and Knowledge Engineering*, 19(1):39–64, 1996.

[JTBM06] L. Jiang, T. Topaloglou, A. Borgida, and J. Mylopoulos. Incorporating Goal Analysis in Database Design: A Case Study from Biological Data Management. In *RE'06*, 2006.

[KB99] Z. Kedad and M. Bouzeghoub. Discovering View Expressions from a Multi-Source Information Systems. In *Proceedings of International Conference on Cooperative Information Systems (CoopIS)*, pages 57–68, 1999.

[KC03] G. Klyne and J. J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Working Draft 23, http://www.w3.org/TR/rdf-concepts, January 2003.

[KCGS93] W. Kim, I. Choi, S. Gala, and M. Scheevel. On Resolving Schematic Heterogeneity in Multidatabase Systems. *International Journal of Distributed Parallel Databases*, 1:251–279, 1993.

[KFNM04] H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen. The Protege OWL Plugin: An Open Development Environment for Semantic Web Applications. In *Proceedings of International Conference on the Semantic Web (ISWC)*, 2004.

[Kim96] R. Kimball. *The Data Warehouse Toolkit*. John Wiley and Sons, 1996.

[KL01] C. Kleiner and U. W. Lipeck. Automatic Generation of XML DTDs from Conceptual Database Schemas. In *GI Jahrestagung (1)*, 2001.

[KLK91] R. Krishnamurthy, W. Litwin, and W. Kent. Languages features for interoperability of databases with schematic discrepancies. In *ACM SIGMOD*, pages 40–49, 1991.

[Klu88] A. C. Klug. On conjunctive queries containing inequalities. *J. of the ACM*, 35(1):146–160, 1988.

[Kol05] P. G. Kolaitis. Schema Mappings, Data Exchange, and Metadata Management. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, 2005.

[KS97] M. Krippendorf and I.-Y. Song. The Translation of Star Schema into Entity-Relationship Diagrams. In *DEXA '97: Proceedings of the 8th International Workshop on Database and Expert Systems Applications*, page 390, Washington, DC, USA, 1997. IEEE Computer Society.

[KS03a] Y. Kalfoglou and M. Scholemmer. Ontology Mapping: The State of the Art. *The Knowledge Engineering Review*, 18(1):1–31, 2003.

[KS03b] Y. Kalfoglou and M. Schorlemmer. IF-Map: An Ontology-Mapping Method Based on Information-Flow Theory. *J. on Data Semantics*, 1:98–127, 2003.

[KX05] Z. Kedad and X. Xue. Mapping Discovery for XML Data Integration. In *Proceedings of International Conference on Cooperative Information Systems (CoopIS)*, 2005.

[LC00] D. Lee and W. W. Chu. Constraint-Preserving Transformation from XML Document Type Definition to Relational Schema. In *Proceedings of International Conference on Conceptual Modeling (ER)*, 2000.

[Len02] M. Lenzerini. Data integration: a theoretical perspective. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 233–246, 2002.

[LMSS95] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, 1995.

[LNE89] J. Larson, S. Navathe, and R. Elmasri. A theory of attribute equivalence in database with application to schema integration. *IEEE trans. Software Eng.*, 15, 1989.

[LRO96] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proceedings of the Twenty-second International Conference on Very Large Databases (VLDB)*, pages 251–262, 1996.

[LS03] L. V. S. Lakshmanan and F. Sadri. Interoperability on XML Data. In *Proceedings of the International Conference on Semantic Web (ISWC)*, 2003.

[LSK96] A. Y. Levy, D. Srivastava, and T. Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems*, 5(2):121–143, Dec. 1996.

[MB+02] J. Madhavan, P. A. Bernstein, et al. Representing and reasoning about mappings between domain models. In *Proceedings of AAAI*, 2002.

[MBDH05] J. Madhavan, P. Bernstein, A. Doan, and A. Halevy. Corpus-Based Schema Matching. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 57–68, 2005.

[MBHR05] S. Melnik, P. A. Bernstein, A. Halevy, and E. Rahm. Supporting Executable Mappings in Model Management. In *Proceedings of ACM SIGMOD*, 2005.

[MBJK90] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: representing knowledge about information systems. *ACM Transaction on Information Systems*, 8(4):325–362, October 1990.

[MBR01] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *Proceedings of the International Conference on Very Large Data bases (VLDB)*, pages 49–58, 2001.

[MFK01] I. Manolescu, D. Florescu, and D. Kossmann. Answering XML Queries on Heterogeneous Data Sources. In *Proceedings of Intl. Conference on Very Large Data Bases (VLDB)*, pages 241–250, 2001.

[MGMR02] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: a versatile graph matching algorithm and its application to schema matching. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 117–128, 2002.

[MH03] J. Madhavan and A. Halevy. Composing Mappings Among Data Sources. In *Proceedings of the International Conference on Very Large Data bases (VLDB)*, 2003.

[MHH00] R. J. Miller, L. M. Haas, and M. A. Hernandez. Schema Mapping as Query Discovery. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 77–88, 2000.

[MIKS96] E. Mena, A. Illarramendi, V. Kashyap, and A. P. Sheth. OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. In *Proceedings of CoopIS'96*, pages 14–25, 1996.

[MIR93] R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. The Use of Information Capacity in Schema Integration and Translation. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*, 1993.

[MM90] V. M. Markowitz and J. A. Makowsky. Identifying Extended Entity-Relationship Object Structures in Relational Schemas. *IEEE Transactions on Software Engineering*, 16(8):777–790, August 1990.

[MRB03] S. Melnik, E. Rahm, and P. A. Bernstein. Rondo: A Programming Platform for Generic Model Management. In *Proc. of ACM SIGMOD*, 2003.

[MS92] V. M. Markowitz and A. Shoshani. Representing Extended Entity-Relationship Structures in Relational Databases: A Modular Approach. *ACM Transactions on Database Systems*, 17(3):423–464, September 1992.

[MvH04] D. L. McGuinness and F. v. Harmelen. *OWL Web Ontology Language Overview*. W3C Recommendation 10, http://www.w3c.org/TR/owl-features, 2004.

[Myl98] J. Mylopoulos. Information Modeling in the Time of the Revolution. *Information Systems*, 23:127–155, 1998.

[MZ98] T. Milo and S. Zohar. Using Schema Matching to Simplify Heterogeneous Data Translation. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*, pages 122–133, 1998.

[NA87] S. B. Navathe and A. M. Avong. Abstracting Relational and Hierarchical Data with a Semantic Data Model. In *Proceedings of International Conference on Entity-Relationship Approach (ER)*, 1987.

[NBM05] A. Nash, P. Bernstein, and S. Melnik. Composition of Mappings Given by Embedded Dependencies. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, 2005.

[NDH05] N. F. Noy, A. Doan, and A. Y. Halevy. Semantic Integration. *AI Magazine*, 26(1):7–9, 2005.

[NM01a] N. F. Noy and M. A. Musen. Anchor-PROMPT: Using Non-Local Context for Semantic Matching. In *Workshop on ontologies and information sharing at IJCAI-2001*, 2001.

[NM01b] N. F. Noy and M. A. Musen. Anchor-PROMPT: Using non-local context for semantic matching. In *IJCAI-2001, Seattle, WA*, 2001.

[NM03] N. F. Noy and M. A. Musen. The PROMPT Suite: Interactive Tools for Ontology Merging and Mapping. *International Journal of Human-Computer Studies*, 59(6):983–1024, 2003.

[PB03] R. A. Pottingerand and P. A. Bernstein. Merging Models Based on Given Correspondences. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*, 2003.

[PH01] R. Pottinger and A. Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB journal*, 10(2-3), 2001.

[PKBT94] J.-M. Petit, J. Kouloumdjian, J.-F. Boulicaut, and F. Toumani. Using Queries to Improve Database Reverse Engineering. In *Proceedings of the 13th International Conference on the Entity-Relationship Approach (ER)*, pages 369–386, 1994.

[PS05] J. Euzenat P. Shvaiko. A Survey of Schema-based Matching Approaches. *Journal on Data Semantics*, 2005.

[PVM+02] L. Popa, Y. Velegrakis, R. J. Miller, M. Hernandes, and R. Fagin. Translating Web Data. In *Proceedings of the International Conference on Very Large Data bases (VLDB)*, pages 598–609, 2002.

[Qia96] X. Qian. Query Folding. In *Proceedings of 12th International Conference on Data Engineering*, pages 48–55, 1996.

[Qui68] M. R. Quillian. Semantic Memory. In *Semantic Information Processing*, pages 227–270. The MIT Press, 1968.

[RB01] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10:334–350, 2001.

[RG02] R. Ramakrishnan and M. Gehrke. *Database Management Systems (3rd ed.)*. McGraw Hill, 2002.

[SdJPA02] P. Sousa, L. Pedro de Jesus, G. Pereira, and F. B. Abreu. Clustering relations into abstract ER schemas for database reverse engineering. *Science of Computer Programming*, 45(2-3):137–153, 2002.

[She95] A. Sheth. Data Semantics: what, where and how. In *Proceedings of the 6th IFIP Working Conference on Data Semantics (DS-6)*, 1995.

[SJ95] I.-Y. Song and T. H. Jones. Ternary Relationship Decomposition Strategies Based on Binary Imposition Rules. In *Proc. of 11th Int'l Conf. on Data Engineering (ICDE '95)*, pages 485–492, March 6-10, 1995.

[SK92] A. P. Sheth and V. Kashyap. So Far (Schematically) yet So Near (Semantically). In *DS-5*, pages 283–312, 1992.

[SL90] A. Sheth and Larson. Federated Database Systems for Managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[SM01] G. Stumme and A. Madche. FCA-Merge: Bottom-up Merging of Ontologies. In *Proceedings of Intl. Conference on Artificial Intelligence (IJCAI)*, pages 225–230, 2001.

[Sou96] C. Soutou. Extracting N-ary Relationships Through Database Reverse Engineering. In *Proceedings of International Conference on Conceptual Modeling (ER)*, pages 392–405, 1996.

[Sou98] C. Soutou. Inference of Aggregate Relationships through Database Reverse Engineering. In *Proceedings of International Conference on Conceptual Modeling (ER)*, pages 135–149, 1998.

[SP94] S. Spaccapietra and C. Parent. View Integration: A Step Forward in Solving Structural Conflicts. *TKDE*, 6(2):258–274, 1994.

[STH+99] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational Database for Querying XML Documents: Limitations and Opportunities. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 1999.

[SY80] Y. Sagiv and M. Yannakakis. Equivalence among relational expressions with the union and difference operators. *J. of ACM*, 27(4):633–655, 1980.

[Ull00] J. D. Ullman. Information integration using logical views. *Theoretical Computer Science*, 239(2):189–210, 2000.

[vdM92] R. van der Meyden. *The complexity of querying indefinite information*. Ph.D. thesis, Rutgers University, 1992.

[VMP03] Y. Velegrakis, R. Miller, and L. Popa. Mapping Adaptation under Evolving Schemas. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*, 2003.

[Wid95] J. Widom. Research Problems in Data Warehousing. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, pages 25–30, 1995.

[Wie92a] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.

[Wie92b] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Compute*, 25(3):38–49, 1992.

[Woo75] W. Woods. What's in a link: Foundations for Semantic Networks. In D. Bobrow and A. Collins, editor, *Representation and Understanding*. Academic Press, 1975.

[WS84] J. A. Wald and P. G. Sorenson. Resolving the Query Inference Problem Using Steiner Trees. *ACM TODS*, 9(3):348–368, 1984.

[WVV+01] H. Wache, T. Vogele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hubner. Ontology-Based Integration of Information - A Survey of Existing Approaches. In *Proceedings of Workshop on Ontologies and Information Sharing in conjunction with the International Joint Conferences on Artificial Intelligence (IJCAI)*, 2001.

[XE03a] L. Xu and D. Embley. Discovering Direct and Indirect Matches for Schema Elements. In *DASFAA*, 2003.

[XE03b] L. Xu and D. Embley. Using Domain Ontologies to Discover Direct and Indirect Matches for Schema Elements. In *Semantic Integration Workshop in ISWC'03*, 2003.

[YMHF01] L. L. Yan, R. J. Miller, L. Haas, and R. Fagin. Data-Driven Understanding and Refinement of Schema Mappings. In *Proceedings of the ACM SIGMOD*, pages 485–496, May 2001.

[YP04] C. Yu and L. Popa. Constraint-based XML query rewriting for data integration. In *Proceedings of ACM SIGMOD*, pages 371–382, 2004.

[YP05] C. Yu and L. Popa. Semantic Adaptation of Schema Mappings when Schemas Evolve. In *Proceedings of the International Conference on Very Large Data bases (VLDB)*, 2005.

[ZM83] Z. Zhang and A. O. Mendelzon. A Graphical Query Language for Entity-Relationship Databases. In *ER'83*, pages 441–444, 1983.