

# A Semantic Approach to Schema Mapping

Yuan An  
University of Toronto  
Ontario, Canada  
yuana@cs.toronto.edu

Alex Borgida  
Rutgers University  
New Jersey, USA  
borgida@cs.rutgers.edu

Renée J. Miller      John Mylopoulos  
University of Toronto  
Ontario, Canada  
{miller,jm}@cs.toronto.edu

## Abstract

We take schema mapping to be the problem of finding an appropriate semantic relationship to load data from a source to a target database, given their schemas. This relationship is expressed in terms of declarative logical expressions. The problem is inherently difficult to automate and previous solutions have proposed algorithms which take as input simple element correspondences between schemas in addition to local schema constraints such as keys and referential integrity. In this paper, we investigate the use of a richer source of information about schemas, namely the presumed presence of semantics for each table expressed in terms of a conceptual model (CM) associated with it. Our approach first compiles each CM into a graph and represents each table’s semantics as a subtree in it. Second, we develop algorithms for discovering subgraphs that are plausible connections between those concepts/nodes in the CM graph that have attributes participating in element correspondences. A conceptual mapping candidate is then a pair of source and target subgraphs which are semantically similar. At the end, these must be converted to database mappings. We offer experimental results demonstrating that, for test datasets drawn from a number of domains, the “semantic” approach outperforms the chase technique which only uses referential integrity constraints, in terms of recall and especially selectivity.

## 1 Introduction

Schema mapping is the problem of finding an appropriate semantic relationship between a source and a target schema. This relationship is expressed in a declarative language. The problem is inherently difficult to automate; therefore, interactive and semi-automatic tools are assumed to be the solution. Such a tool often employs a two-phase paradigm: first, specify some simple correspondences between schema elements; then derive plausible declarative mapping expressions for users to select from. Clio [13, 14]

is the best known such tool.

For example, to find the mapping from schema  $S$  to schema  $T$  in Figure 1, one specifies/discovered the correspondences  $v_1, v_2, v_3$ , and  $v_4$ . This indicates, for instance, that the `addr` column of the `address` table corresponds to the `addr` column of the `personnel` table. Clio is able to generate a mapping formula that creates a `personnel` tuple by joining together a `professor` tuple and an `address` tuple. This mapping is suggested by the presence of the foreign key constraint ( $r$ ) between the relations. Depending on other constraints, one may choose an outer-join, rather than a join to avoid losing information.

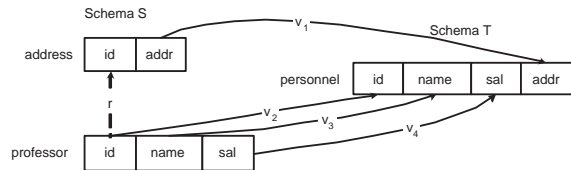


Figure 1: Input to Schema Mapping Discovery

Schema mapping lies at the heart of a variety of data management problems. Effective and labor-saving tools are always in great demand by data integration practitioners. Currently, progress has been made in automatically specifying element correspondences [15, 10, 12, 6]. In this paper, we focus on the second problem, that of deriving plausible declarative mapping expressions starting from element correspondences. Given a set of element correspondences from the source to the target schemas<sup>1</sup> (in this paper, these relate column names), the essential step in deriving mappings is to find some way of “connecting semantically” the elements in the source and in the target schemas. The current best solution [14] uses a variant of the chase algorithm on foreign key references to assemble “logically connected elements”. However, as shown by the motivating examples in next subsection, this technique sometimes does not produce directly the most natural semantic connections. Moreover, the space

<sup>1</sup>The correspondences are directed from the source to the target only in the sense that data from the source columns is intended to contribute to data that will appear in the target column.

of all possible mappings may be very large, and it would be desirable to provide the tool with an ability to rank-order them.

In this paper, we investigate the potential benefits of a complementary approach, which assumes that the specific *semantics* of database schemas are also available as input. Data semantics amounts to establishing and maintaining a relationship between an information system (model) and its intended subject [5]. To capture the semantics of a database schema, we will use a conceptual model (abbreviated as CM). 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, maintaining the EER schema and the mapping between the EER schema and the relational schema needs limited effort<sup>2</sup>. In addition, we have recently developed a tool [2, 3] to recover the semantic mapping between a legacy database schema and an existing CM that covers the same domain of discourse as the database.

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 elements, which have proven to be so useful in Clio.

## 1.1 Terminology and Motivating Examples

We next clarify the input/output of our program and provide motivation and terminology using illustrative (rather than exhaustive) examples.

**Example 1.1:** Consider the source relational schema given in the upper part of Figure 2. It contains five tables: `person(pname)`, `writes(pname, bid)`, `book(bid)`, `soldAt(bid, sid)`, and `bookstore(sid)`. The underlined column name(s), such as `pname`, indicates the primary key of the 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 recovers 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)`, 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 this case, the associations are intuitively obvious. We provide a formal definition in Section 2). We assume that a CM

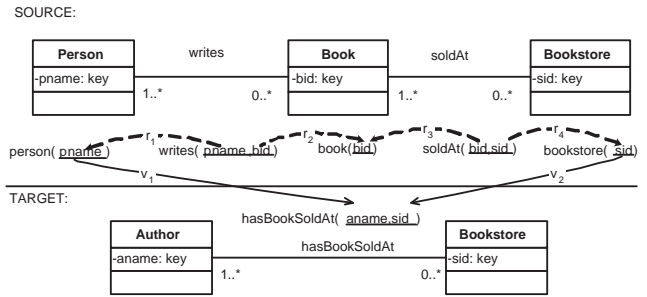


Figure 2: Schemas, CMs, RICs, and Correspondences

contains, as usual, *entity sets (classes)*, *relationships between classes (properties)*, *attributes*, and *cardinality constraints* imposed on the participation of classes in relationships. We take a binary relationship to link its participating classes in a specific direction. For example, Person is linked to Book by `writes`, while Book is linked to Person by `writes-`, which is the inverse of `writes`. We use UML’s notation for cardinality constraints, where *min..max* specifies lower and upper bounds on the number of range objects related to a single domain object. Thus, `0..*` means “no constraint”, `1..*` means “total participation”, while `...1` indicates that each domain instance *functionally determines* the other participating instance in *R*.

To encode constraints for identifying objects, we need a special **key** notation to indicate (collections of) attributes that act as identifiers of entities.

There are well-known methodologies for designing logical database schemas from a CM (e.g., ER diagrams). We call such methodologies *er2rel designs* (e.g. [11]). Essentially, the *er2rel* design maps each class/entity to a class/entity table and each relationship to a relationship table. In addition, it also permits merging table  $T_1$  into table  $T_2$  when the key of  $T_1$  has an RIC to the key of  $T_2$ ; such a merge reduces the need for joins, at the cost of possibly introducing null values in some columns of  $T_1$ .

Continuing with our example, a *target schema* is also given in the lower part of Figure 2. 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. As in [14], we use the simplest form of correspondences, relating pairs of column names in the source and target. Figure 2 show two correspondences using solid lines:  $v_1$ , 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  $\leftrightarrow$  hasBookSoldAt.aname`.

**Current Solution** One approach takes as input the source schema, the target schema, the RICs, and the correspondences. To generate a declarative mapping expression,

<sup>2</sup>In [3], we have shown how to do this formally for standard designs.

the “chase solution” employs an extension of the *relational chase* algorithm to first assemble logically connected elements into so-called “logical relations”, before connecting them with correspondences. The result in this case is as follows: Let the four RICs in Figure 2 be:  $r_1:\text{writes.pname} \subseteq \text{person.pname}$ ;  $r_2:\text{writes.bid} \subseteq \text{book.bid}$ ;  $r_3:\text{soldAt.bid} \subseteq \text{book.bid}$ ;  $r_4:\text{soldAt.sid} \subseteq \text{bookstore.sid}$ .

The result of chasing the table `writes(pname,bid)` using RIC  $r_1$  can be represented by the algebraic expression:

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

Applying the chase again to  $S_1$  using  $r_2$ , we obtain the expression:

$$S_2:\text{person(pname)} \bowtie \text{writes(pname, bid)} \bowtie \text{book(bid)}.$$

Since no further chase steps can be applied (i.e.,  $S_2$  cannot be expanded further)  $S_2$  is a *logical association*. Likewise, the result of chasing the table `soldAt(bid,sid)` using  $r_3$  and  $r_4$  is the logical relation:

$$S_3:\text{book(bid)} \bowtie \text{soldAt(bid,sid)} \bowtie \text{bookstore(sid)}.$$

In the target, a logical relation is:

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

To interpret the correspondences  $v_1$  and  $v_2$ , Clio’s algorithm looks at each pair of source and target logical relations. For each such pair, it computes source-to-target mappings driven by the correspondences that are *covered* by the pair. For example, the pair  $\langle S_2, T_1 \rangle$  covers  $v_1$ , and the pair  $\langle S_3, T_1 \rangle$  covers  $v_2$ . So the mappings are actually written as  $\langle S_2, T_1, v_1 \rangle$  and  $\langle S_3, T_1, v_2 \rangle$ . The algorithm will then generate the following two candidate *declarative mapping expressions*, which have the form of so-called “tuple generating dependencies”<sup>3</sup>:

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

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

In  $M_1$  and  $M_2$ , existentially quantified variables need to be realized as labeled nulls. Note that if outer-joins are used in the algebraic expressions  $S_2$  and  $S_3$ , the following expressions are also candidate mappings:

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

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

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

**Alternate Solution** We believe that a more natural mapping expression in this case would be:

$$M_5:\forall \text{pname} \forall \text{bid} \forall \text{sid} (\text{person}(\text{pname}) \wedge \text{writes}(\text{pname}, \text{bid}) \wedge \text{soldAt}(\text{bid}, \text{sid}) \wedge \text{bookstore}(\text{sid}) \rightarrow \text{hasBookSoldAt}(\text{pname}, \text{sid}))^3.$$

The mapping basically says that a person and a bookstore together are mapped to the target if the person writes a

<sup>3</sup>Note that even if all we are interested in is loading data from the source to the target schema, the existentially quantified variables require the introduction of different Skolem functions in key fields (see [14]), so the two earlier mappings cannot produce tuples that can be merged. That is why it is important to create larger logical relations.

book and the book is sold at the bookstore. Now looking into the semantics of the schemas, we observe that there is indeed a connection from the Person class to the Bookstore class, i.e., the composition of `writes` and `soldAt`. Furthermore, the cardinality constraints of the composed connection (many to many) are compatible with that of the target relationship `hasBookSoldAt`. Therefore, the pair  $\langle \text{writes} \circ \text{soldAt}, \text{hasBookSoldAt} \rangle$  of connections in the CMs is a plausible mapping from the source CM to the target CM if Person in the source corresponds to Author in the target, and Bookstore in the source corresponds to Bookstore in the target. With this information, the semantics of the schemas may help us to generate  $M_5$  directly.  $\square$

The use of the semantics of schemas is also important in the following example.

**Example 1.2:** Most conceptual modeling languages support the notion of subclasses/superclasses connected by ISA relationships. In addition, there may be constraints that apply to the ISA relationship, such as *disjointness*, specifying that a certain collection of subclasses have non-overlapping extents.

In the example we are to consider, the CM associated with the source schema in Figure 3 contains a superclass Employee and two subclasses Engineer and

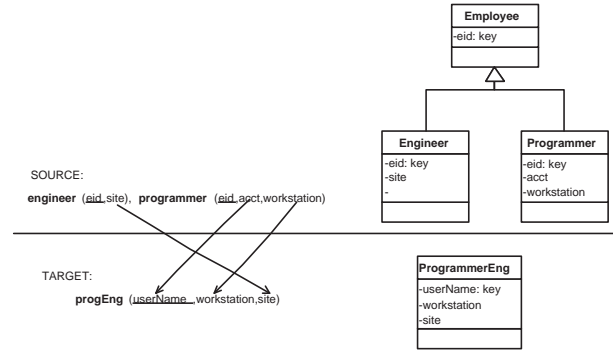


Figure 3: Using Subclass Semantics in CM

Programmer. Suppose the ISA hierarchy is total and overlapping, which means that each Employee must be an Engineer or Programmer or both. A possible way of translating the CM into a relational schema by the `er2rel` design is two separate tables: one for Engineer, another for Programmer, both using the inherited identifier `eid` as their primary keys. Given a target schema containing a single table tracking those programmers who are also engineers, the following mapping

$$M:\forall \text{acct} \forall \text{eid} \forall \text{ws} \forall \text{site} (\text{programmer}(\text{eid}, \text{acct}, \text{ws}) \wedge \text{engineer}(\text{eid}, \text{site}) \rightarrow \text{progEng}(\text{acct}, \text{ws}, \text{site})),$$

is expected for interpreting the correspondences shown as the solid lines in Figure 3. Notice that the mapping  $M$  will not be discovered by the chase algorithm (i.e., no chase can apply). However, the CM associated with the

source schema indicates that there is a connection between Engineer and Programmer through their common superclass Employee. Using this semantics, it is possible to generate the mapping  $M$  directly. On the other hand, note that if the two subclasses were disjoint, we should avoid generating the above mapping. This is feasible if the disjointness constraint is represented in the CM. However, we believe that detecting disjointness cannot be achieved by merely looking at logical schemas.

Moreover, the semantics may also help us to downgrade/eliminate incompatible pairs. For example, suppose that the source contains tables `household(hid,address)` and `person(pid,name,age)`, as well as separate tables for three relationships between them: `memberOf(pid,hid)`, `headOf(pid,hid)`, and `pastorOf(pid,hid)`, where the `pid` and `hid` columns of `memberOf/headOf/pastorOf` are foreign keys referencing `person.pid` and `household.hid`, respectively. The formal semantics indicate that `memberOf` and `headOf` are **part-of** relationships (i.e., removing a household cascades to the removal of parts) in contrast to `pastorOf`; while `headOf` and `pastorOf` are functional from households to persons (i.e., the cardinality upper bound is 1), in contrast to `memberOf`. Suppose that on the target side, we have tables `famiglia(fid,addr)`, `persona(pid,nome)` and some other table `foo(pid,fid)`, where `foo.pid` and `foo.fid` are foreign keys referencing `persona.pid` and `famiglia.fid`, respectively. The only correspondences are between the externally visible information: `household.address`  $\rightsquigarrow$  `famiglia.addr` and `person.name`  $\rightsquigarrow$  `persona.nome`. According to chase approach, there are three candidate mappings, from `household`  $\bowtie$   $\{memberOf|headOf|pastorOf\}$   $\bowtie$  `person` respectively to `famiglia`  $\bowtie$  `foo`  $\bowtie$  `persona`, and there is nothing to distinguish them. However, if we know from the semantics that `foo` is not a **part-of** relationship, then we would expect it unlikely that `memberOf/headOf` would have lead to mappings. Similarly, if `foo` had functional semantics, then we would have considered it unlikely that `memberOf` was used in the source, since this would likely lead to the violation of the upper bound 1 constraint in some cases.  $\square$

The rest of the paper presents our principled approach to the problem of schema mapping. Section 2 proposes a representation of table semantics. Section 3 presents element correspondences and schema mapping representation. Section 4 illustrates the algorithm for mapping generation. Section 5 evaluates the approach using a comprehensive set of experiments. Section 6 discusses related work. Finally, Section 7 presents conclusions and points to possible future directions.

## 2 Representing the Semantics of Schemas

We shall represent a given CM using 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 classes). We construct the CM graph from a CM as follows: We create a class node labeled with  $C$  for each class  $C$ , and an edge labeled with  $p$  from the class node  $C_1$  to the class node  $C_2$  for each binary relationship set  $p$  linking  $C_1$  to  $C_2$ ; for each such  $p$ , there is also an edge in the opposite direction for its inverse, referred to as  $p^-$ .

Note that we will deal with n-ary relationships, relationships with attributes, and so-called higher order relationships (which relate relationships themselves) in Section 4.2 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.

For each attribute  $f$  of a class  $C$ , we create a separate attribute node, whose label is  $f$ , and add an edge labeled with  $f$  as well from the node  $C$  to the attribute node.<sup>4</sup> For each **isa** edge from a subclass  $C_1$  to a superclass  $C_2$ , we create an edge labeled with **isa** from the node  $C_1$  to the node  $C_2$ . For the sake of succinctness, we use UML notation to represent a CM graph. Note that in such a diagram, instead of drawing separate attribute nodes, we place the attributes inside the rectangle nodes; and relationships and their inverses are represented by a single undirected edge. The presence of such an undirected edge, labeled  $p$ , between classes  $C$  and  $D$  will be written in text as  $\boxed{C} \text{ --- } p \text{ --- } \boxed{D}$ . 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  $\boxed{C} \text{ --- } p \text{ -> --- } \boxed{D}$ . Each **isa** relationship has cardinality 1..1, and its inverse, **mayAlsoBe**, has cardinality 0..1.

In this paper, the semantics of a table is represented by a subtree in a CM graph. We call such a subtree *semantic tree* (or *s-tree*), where columns of the table associate uniquely with attribute nodes of the s-tree. This representation of table semantics was presented in [4], and corresponds to a kind of conjunctive formula which can be derived from the s-tree. The encoding uses unary predicates for classes, binary predicates for attributes, and binary predicates for binary relationships. For relational schemas, we use a formula of the form  $T(X) \rightarrow \exists Y. \Phi(X, Y)$  to represent the semantics of table  $T$ , where  $T$  is a table with columns  $X$  (which become arguments to its predicate), and  $\Phi$  is a conjunctive formula over predicates representing an s-tree. The encoding introduces a new variable for every node in the tree, and proceeds recursively (see [4]). For example, source table `writes(pname,bid)` in Figure 2, whose semantics is represented by the s-tree consisting of nodes `Person` and `Book` connected by edge `writes`, has logical semantics

<sup>4</sup>Unless ambiguity arises, we say “node  $C$ ”, when we mean “class node labeled  $C$ ”.

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

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

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 [4]). So, for example, the semantics of table `pers(pid,name,age,spousePid)` is represented by a graph with two nodes, `Person` and `Personcopy1`, connected by edge `hasSpouse`. And an additional column, `pers.bestFriendPid`, would require an additional node, `Personcopy2`, 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.

It is in fact possible to take an arbitrary formula of the form  $T(X) \rightarrow \exists Y.\Phi(X, Y)$  and convert it into a semantic graph (though possibly no longer a tree): (i) for every binary atom  $p(x, y)$ , add, without duplication, unary restrictions  $D(x)$  and  $R(y)$  where  $p$  was connecting  $D$  and  $R$  in the CM. (ii) Any time the same variable  $x$  occurs as the argument of two concepts  $C$  and  $D$ , replace  $x$  by a new variable  $z$  as the argument of  $D$ , and of every binary relation  $q(x, w)$  leaving  $D$ ; when such variables are introduced, add clauses  $(x = z)$ , unless  $C$  isa  $D$ , in which case add clause  $isa(x, z)$ . For selection queries with constants, which appear as arguments in atoms (e.g., `hasAge(x, 9)`), the attribute node can be labeled with a numeric value. As a result, every conjunctive query  $\Phi(X, Y)$  describing the semantics of a table can be represented as an s-tree, with some additional equality edges connecting nodes.

The previous study [4] also associates additional notions with the semantics of a table  $T$ . The first is an *anchor* which is the central object in the s-tree from which  $T$  is derived, in case `er2rel` was used. For example, if  $\tau(\underline{c}, d)$  was derived from a functional relationship  $\boxed{C} \text{ ---p--- } \boxed{D}$ , then  $C$  is the anchor of table  $T$ . The second is 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 the additional notions can be found in [4].

### 3 Element Correspondences and Schema Mapping Representation

As mentioned before, in this paper, we study the problem of deriving plausible declarative mapping expressions starting from element correspondences. Here, an element correspondence links a source column to a target column. For example, the correspondence  $v_1$  links the source col-

umn `person.pname` to the target column `hasBookSoldAt.aname` in Example 1.1.

For a set  $\mathcal{L}$  of correspondences between a schema  $S$  and a schema  $T$ , we use  $\mathcal{L}(S)$  and  $\mathcal{L}(T)$  to denote the sets of columns linked by  $\mathcal{L}$  in  $S$  and  $T$ , respectively. The goal of schema mapping is to find an algebraic expression connecting columns in  $\mathcal{L}(S)$  and an algebraic expression connecting columns in  $\mathcal{L}(T)$  such that the pair “interprets”  $\mathcal{L}$ . The interpretation depends on the intention underlying the correspondences. As we have indicated, automatic tools are available for specifying element correspondences. Their underlying principles are finding *similar* pairs in schemas based on linguistic, structural, and statistical information. Assuming that  $\mathcal{L}$  specifies two sets of similar table columns, that is, every column  $c \in \mathcal{L}(S)$  has a similar counterpart column  $d \in \mathcal{L}(T)$  in terms of modeling a real-world subject matter, we will craft our schema mapping approach to find a pair of “similar” relational expressions in terms of their connections for interpreting the correspondences  $\mathcal{L}$ .

Since there are considerable complexities involving issues such as joins/outer-joins, the generation of Skolem constants, etc. which do not differ from [14], we content ourselves with presenting mapping candidates as 3-tuples  $\langle E_1, E_2, \mathcal{L} \rangle$ , where  $E_1$  and  $E_2$  are relational expressions (in algebra or conjunctive query form) for the source and target, respectively, and  $\mathcal{L}$  is a set of correspondences that are covered by the pair of expressions.

Our approach will be to find mapping candidates  $\langle D_1, D_2 \rangle$  at the *conceptual level*, where  $D_1$  and  $D_2$  are subgraphs of the corresponding CM graphs. These candidates are then converted to relational mappings by a technique akin to query rewriting using views.

## 4 Generating Mapping Candidates

Given a relational schema  $S$  associated with a CM  $\mathcal{G}_S$  through the semantic mapping  $\Sigma_S$  and a relational schema  $T$  associated with a CM  $\mathcal{G}_T$  through the semantic mapping  $\Sigma_T$ . Given also  $\mathcal{L}$ , a set of correspondences linking a set  $\mathcal{L}(S)$  of columns in  $S$  to a set  $\mathcal{L}(T)$  of columns in  $T$ .

As shown earlier, the semantic mappings relate each table in the schemas to an s-tree in the respective CM graphs, associating with each column a class node in the graph through the bijective associations from columns to attribute nodes. Consequently, the set  $\mathcal{L}(S)$  of columns gives rise to a set  $\mathcal{C}_S$  of “marked” class nodes in the graph  $\mathcal{G}_S$ . Likewise, the set  $\mathcal{L}(T)$  gives rise to a set  $\mathcal{C}_T$  of “marked” class nodes in the graph  $\mathcal{G}_T$ . We call the s-trees associating with tables that have columns participating in  $\mathcal{L}$  *pre-selected s-trees*. Subsequently, our approach consists of two major steps: (1) finding 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 “similar” — by analogy with [14], we call these

“logical subgraphs”; (2) restoring the attribute nodes used to identify the class nodes and translating  $D_1$  into an algebraic expression  $E_1$  and  $D_2$  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  $\mathcal{L}$  or a subset of  $\mathcal{L}$ .

In the rest of this section, we first describe the logical subgraphs for different situations and develop algorithms for discovering them using various examples. Then, we propose an approach to translating a logical subgraph into an algebraic expression using techniques of rewriting queries using views.

#### 4.1 Basic Conceptual Model

The basic CM contains classes, binary relationships, and ISA relationships. All other relationships are reified in the CM graph, and we delay their treatment to the next subsection.

For a set  $\mathcal{C}_S$  of nodes in the source CM graph  $\mathcal{G}_S$  and a set  $\mathcal{C}_T$  of nodes in the target CM graph  $\mathcal{G}_T$ , there are many ways to connect nodes in  $\mathcal{C}_S$  and nodes in  $\mathcal{C}_T$ . We propose to systematically explore the information encoded in the correspondences and the semantic mappings for discovering the pair of similar logical subgraphs. First, a node  $v \in \mathcal{C}_S$  corresponds to a node  $u \in \mathcal{C}_T$  when  $v$  and  $u$  have attributes that connect to corresponding columns via the semantic mappings. Second, we take the following observations/conditions into consideration: (i) For a pair of nodes  $(v_1, v_2)$  in  $\mathcal{C}_S$  and a pair of nodes  $(u_1, u_2)$  in  $\mathcal{C}_T$ , if  $v_1$  corresponds to  $u_1$  and  $v_2$  corresponds to  $u_2$ , if there is to be a link between  $v_1$  and  $v_2$  then it should be “semantically similar” or at least “compatible” to the link between  $u_1$  and  $u_2$ ; for example, a source link with lower bound 2 of the cardinality for  $v_2$  is not compatible with a target link with upper bound 1 of the cardinality for  $u_2$ . (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 logical subgraphs use edges from the pre-selected s-trees. (iii) To the extent that there are choices available, we want the logical subgraph to represent “intuitively meaningful queries”. In relational database there appears to be consensus that this requires the joins in the query to be lossless. (See more below.) (iv) All things being equal, we want the logical subgraph to be compact – as per Occam’s principle.

Observation (iv) leads to our first heuristic for connecting nodes: if there are no other constraints, shortest paths in the conceptual graph are preferred. In the relational context, observation (iii) led to the use of the chase operation on RICs, since lossless joins are considered a useful guideline for creating semantically meaningful queries. Previous research on graphical querying of Entity Relationship diagrams [18] has identified that *functional trees* correspond to

“lossless joins”. Formally, a functional tree  $\mathcal{F}$  containing a set of nodes  $\{v_1, v_2, \dots, v_n\}$  is a tree with a root  $u$  such that the path from  $u$  to each  $v_i (\neq u), i = 1..n$  is functional. (Such a tree is formally a Steiner tree: a spanning tree allowed to pass through additional nodes in order to reach marked nodes.) The preference for functional trees is motivated by the fact that functional associations determine functional dependencies, and hence the application of the er2rel design to a functional tree give rises to a set of relational tables whose join is lossless. Combining the two previous observations, we are led to seek *minimal functional trees* containing, as a subset, the nodes in  $\mathcal{C}_S$  ( $\mathcal{C}_T$ ), unless there are other preemptive conditions. Minimality means that tree  $\mathcal{F}$  does not have a subtree which is also a functional tree containing  $\{v_1, v_2, \dots, v_n\}$ . Interestingly, [16] also considered the problem of querying ER diagrams, and 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.

We now begin to present the algorithm. We will always start from the target side. That is, supposing we have a logical subgraph in the target CM graph, we aim to find a “similar” logical subgraph in the source. There are two subcases:

Case A: The target logical subgraph  $D_2$  is given, e.g., it is the s-tree associated with a single table.

Case B: The target logical subgraph is to be constructed itself.

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

**Example 4.1** Consider the target and source CM graphs in Figure 4. The target tree of relational table `proj(pnum, dept, emp)` can be expressed as the formula:

$$\begin{aligned} \Sigma_1: T:\text{proj}(pnum, dept, emp) \rightarrow \mathcal{O}:\text{Proj}(x), \mathcal{O}:\text{pid}(x, pnum), \\ \mathcal{O}:\text{Dept}(y), \mathcal{O}:\text{did}(y, dept), \mathcal{O}:\text{Emp}(z), \\ \mathcal{O}:\text{eid}(z, emp), \mathcal{O}:\text{hasDept}(x, y), \mathcal{O}:\text{hasSup}(x, z). \end{aligned}$$

There are a number of source tables whose semantics are given as s-trees of the source CM graph, including `control(proj,dept)` and `manage(dept,mgr)`. Their semantics are:

$$\begin{aligned} \Sigma_2: T:\text{control}(proj, dept) \rightarrow \mathcal{O}:\text{Project}(x), \mathcal{O}:\text{pid}(x, proj), \\ \mathcal{O}:\text{Department}(y), \mathcal{O}:\text{did}(y, dept), \mathcal{O}:\text{controlledBy}(x, y). \\ \Sigma_3: T:\text{manage}(dept, mgr) \rightarrow \text{Employee}(y), \mathcal{O}:\text{eid}(y, mgr) \\ \mathcal{O}:\text{Department}(x), \mathcal{O}:\text{did}(x, dept), \mathcal{O}:\text{hasManager}(x, y). \end{aligned}$$

Suppose a user has specified the correspondences  $v_1:\text{control.proj} \leftrightarrow \text{proj.pnum}$ ,  $v_2:\text{control.dept} \leftrightarrow \text{proj.dept}$ , and  $v_3:\text{manage.mgr} \leftrightarrow \text{proj.emp}$ . In Figure 4, the correspondences are lifted to correspondences between the associated class nodes.

Notice that the target logical subgraph is an *anchored s-tree*, where the anchor is Proj, and the path from the anchor

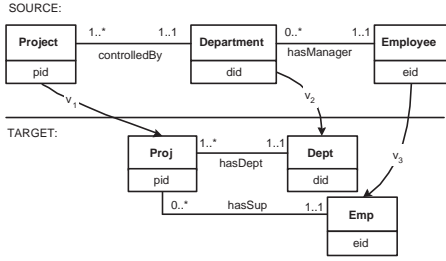


Figure 4: Input to Example 4.1

to every other node is functional. This leads us to believe that a “similar” logical subgraph in the source should be a functional tree with a root corresponding to the anchor.

**Case A.1** Suppose we find a node in the source corresponding to the root Proj in the target, in this case Project. Then 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 the observation (ii) above motivates us to use edges in the pre-selected trees as much as possible, to fulfill this, the edges in pre-selected trees do not contribute to the cost of paths. By doing so, a “shortest path” will contain edges in the pre-selecte trees as much as possible, and cover as many nodes with correspondences as possible. At last, we choose the functional trees of minimal cost as the logical subgraphs in the source. In this example, the tree `Project` `---controlledBy---` `Department` `---hasManager---` `Employee` is the logical subgraph in the source that matches the target logical subgraph.

**Case A.2** If a user only specifies correspondences  $v_2$  and  $v_3$  ( $v_1$  is missing), then we no longer have the information to find the corresponding root in the source; we are nonetheless seeking an logical subgraph in the source that is a functional tree. In this case, we must find all functional trees in the source each of which contains the nodes in  $C_S$ . 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.

Suppose that the nodes in  $C_S$  are not covered by a single (minimal) functional tree in the source CM graph. Then we attempt to connect the nodes as follows. 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 rest unconnected. Consequently, the correspondences will be split among the tree and the remaining unconnected nodes. In Case A.2, we find the trees covering different subsets of the nodes so that the entire set would be covered by several trees with different roots. Accordingly, a pair of trees consists of a source tree covering subsets of nodes identified by the correspondences and the target tree. Consequently, the coverage of the correspondences is split among all the pairs. The same treatment applies to the rest cases as well.  $\square$

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

**Example 4.2** The target tree in Figure 3 of Example 1.2 consists of a single node ProgrammerEng. Using the correspondences `programmer.acct` `---<=>` `programmerEng.userName`, `programmer.workstation` `---<=>` `programmerEng.workstation`, and `engineer.site` `---<=>` `programmerEng.site`, we identify two nodes, Engineer and Programmer, in the source corresponding to ProgrammerEng. To connect the two source nodes, we use key information. The class Programmer has an attribute corresponding to the key userName of the target class ProgrammerEng. Therefore, we search for a functional tree rooted at Programmer, which has the attribute corresponding to userName. As a result, the tree `Programmer` `---isA---` `Employee` `---mayAlsoBe---` `Engineer`, where `mayAlsoBe` is the inverse of `isA`, is returned. Note that there is no disjointness constraint specified between the subclasses. Otherwise, the connection is prohibited.

However, if there is no class 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 4.1.  $\square$

**Case B.** Finally, we consider the case where the target logical subgraph  $D_2$  starts out as a forest consisting of a set of pre-selected s-trees.

The guideline for connecting a set of pre-selected s-trees is the minimal functional tree. Given a set of pre-selected s-trees  $D=\{d_1, d_2, \dots, d_n\}$  in a CM graph  $\mathcal{G}$ . Let the set of nodes  $V=\{v_1, v_2, \dots, v_m\}$  be the nodes associated with a set of correspondences. We know that each  $v_i, i = 1..m$  belongs to some trees  $d_{i_j}, i_j \in \{1..n\}$ . We construct all minimal functional trees containing nodes in  $V$  using the edges in the pre-selected s-trees as much as possible. Consequently, we construct a set of minimal functional trees in the target. Accordingly, we can construct a set of minimal functional trees in the source.

From the sets of minimal functional trees in the source and target, we form a pair of logical subgraphs along with the correspondences covered by the pair. A heuristic used in forming the pairs is similar to that in Case A, i.e., matching

up the roots of the pair of trees if possible. However, if we cannot find a source tree with the corresponding root to the root of a target tree, then the target tree can be paired up with any source tree. Finally, all such pairs are returned as mapping candidates.

## 4.2 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 functional “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 5.) Such reified relationship nodes will be indicated in our text by tagging their name with  $\diamond$ , 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 semantic mapping formulas for tables, 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 5, then the semantic mapping formula is specified as follows:

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

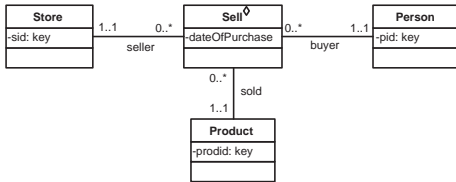


Figure 5: Reified Relationship Node

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 works\_in.department can still be recognized in reified form as `Employee` ---worker---> `WorksIn` ---place---> `Department`.

When a reified relationship node appears in a CM graph, we have several adjustments in the mapping algorithm.

First, a path of length 2 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 target trees rooted at reified relationships induces *preferences* for similarly rooted (minimal) functional trees in the source. This includes the anchor being many-to-many, many-to-one or one-one (distinguished by the cardinality restrictions on the role inverses, as in the WorksIn example above), the number of roles (exact arity), or subclass relationship to top-level ontology concepts such as PartOf $\diamond$ .

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 an N:M relationship between persons and cities where the store is located. Therefore in seeking matches for (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 that are not functional in either direction.

**Example 4.3** The solution to the problem in Example 1.1 is then obtained as follows. The target s-tree in Figure 2 is a many-to-many relationship, which our algorithm represents as a reified relationship with anchor HasBookSoldAt. To find a matching logical subgraph 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. Using a single reified relationship as an anchor, extended by functional paths from the roles corresponds to lossless joins with the table representing the root, and hence are preferred. In this case, no such path can be found. Then we look for longer paths, minimizing the number of few lossy joins, by minimizing the number of direction reversal changes along each path. In this case we get the path from Person to Bookstore through writes $\diamond$ , Book, and soldAt $\diamond$ .  $\square$

## 4.3 Obtaining Algebraic Expressions

The final mapping expression includes a pair of algebraic expressions using the tables in the input relational schemas only. Therefore, we need to translate the discovered logical subgraphs in the CM graphs into algebraic expressions over the database schemas. This process is a case of answering queries using views [7]. To draw the analogy, we take an entire CM as a collection of primitive relations/predicates for its concepts, attributes and properties; the semantics of each input table is a view definition over these predicates;

and a discovered subgraph is a query over the CM relations.

The first step of the translation is to express the discovered subgraph into a query using CM predicates. The encoding algorithm proposed in [4] can be used for this purpose. Here we use the following example to illustrate the encoding process.

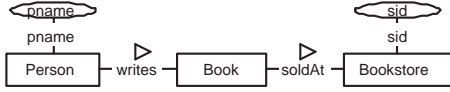


Figure 6: A Discovered Tree over a CM Graph

**Example 4.4** Figure 6 is a fully specified tree in the source CM of Example 1.1. (For simplicity of presentation, we have reduced the reified binary relationships to simple associations. Our algorithm would work correctly even with the reified versions.) Notice that the attributes which associate with the table columns linked by the correspondences are also shown on the tree. The triangles indicate the associating directions of the 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 make up the answer tuples. Assigning a name  $ans$  to the query, we obtain

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

Given the set of semantic mappings for the tables, we can apply techniques for rewriting a query using views to rewrite  $q$  above to a new query  $q'$  which only mentions the tables in the relational schema. The rewritten query  $q'$  is maximally-contained in  $q$  and must mention tables that have columns linked by the correspondences.

**Example 4.5** We have proposed in [4] an ad-hoc approach to deriving inverse rules for each predicate in the CM, in terms of the tables in the relational schema. An essential problem that needs to be resolved here is that unique internal object identifiers (e.g., the arguments  $x$  of predicates like  $\text{Person}(x)$ ) used in the CM, are not directly available in the relational tables. These are formally converted to Skolem functions, giving rise to formulas such as

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

when inverting a semantic specification such as

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

$$\mathcal{O}:\text{hasName}(x, \text{pname}), \mathcal{O}:\text{hasAge}(x, \text{age}).$$

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 Section 2) in order to “unify” the various Skolem functions. So  $f(\text{pname}, \text{age})$  would be replaced by  $\text{pname}$ , if we knew that  $\text{pname}$  is the key of the table.

As a result, using the set of inverse rules, we can rewrite the query  $q$  to queries that mention tables only. In our case, these include the following:

$$q'_1: ans(v_1, v_2) :- T:\text{writes}(v_1, y), T:\text{soldAT}(y, v_2).$$

$$q'_2: ans(v_1, v_2) :- T:\text{person}(v_1), T:\text{writes}(v_1, y), T:\text{book}(y), \\ T:\text{soldAT}(y, v_2), T:\text{bookstore}(v_2).$$

$$q'_3: ans(v_1, v_2) :- T:\text{person}(v_1) T:\text{writes}(v_1, y), \\ T:\text{soldAT}(y, v_2), T:\text{bookstore}(v_2).$$

Since  $q'_1$  does not mention tables  $\text{person}(\text{pname})$  and  $\text{bookstore}(\text{sid})$  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.

Note that the semantics of tables in [4] actually consider nulls in tables, and outer-joins of predicates in the CM, in order to deal with the relationships that have cardinality lower bound 0. Taking these into account complicates the rewriting process.  $\square$

#### 4.4 The Algorithm

We now summarize the core of the mapping discovery algorithm that has been illustrated throughout this section. .

**Specification:** Semantic Mapping Discovery.

**Input:** A source schema  $S$  associating with a CM  $\mathcal{G}_S$  through the semantic mapping  $\Sigma_S$ ; a target schema  $T$  associating with a CM  $\mathcal{G}_T$  through the semantic mapping  $\Sigma_T$ ; a set of correspondences  $\mathcal{L}$  between a set of columns  $\mathcal{L}(S)$  in  $S$  and a set of columns  $\mathcal{L}(T)$  in  $T$ .

**Output:** Mapping candidates  $\langle E_1, E_2, \mathcal{L}_M \rangle$ , where  $E_1$  and  $E_2$  are algebraic expressions over  $S$  and  $T$ , respectively, and  $\mathcal{L}_M \in \mathcal{L}$  is a set of correspondences covered by  $\langle E_1, E_2 \rangle$ .

**Steps:**

1. Find a set of “marked” class nodes  $\mathcal{C}_S$  in  $\mathcal{G}_S$  using  $\mathcal{L}(S)$  and  $\Sigma_S$  and a set of “marked” class nodes  $\mathcal{C}_T$  in  $\mathcal{G}_T$  using  $\mathcal{L}(T)$  and  $\Sigma_T$ .
2. Find a logical subgraph  $D_2$  connecting nodes in  $\mathcal{C}_T$ :  $D_2$  could be: (i) a given s-tree; (ii) a minimal functional tree.
3. Find a logical subgraph  $D_1$  connecting nodes in  $\mathcal{C}_S$  such that  $D_1$  is similar to  $D_2$ : The similarity is suggested by: (i) a (minimal) functional tree matching a given anchored s-tree; (ii) the same shape non-anchored tree using non-functional paths connecting nodes; (iii) two minimal functional trees; (iv) other semantics like **part-of**.
4. Restore attribute nodes associated with correspondences to the logical subgraphs and encode  $D_1$  and  $D_2$  into queries; rewrite the queries into algebraic expression  $E_1$  and  $E_2$  using tables in the input schemas.

- Identify the correspondences  $\mathcal{L}_M$  covered by  $\langle E_1, E_2 \rangle$ ; return  $\langle E_1, E_2, \mathcal{L}_M \rangle$ .

## 5 Experimental Results

We now report on experimental results that demonstrate the performance of the proposed approach. We show that this approach works reasonably in a number of cases, and in general has better results than chase approach. Furthermore, we claim that our techniques are especially effective on schemas derived from CMs with rich structures. 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 collected a number of relational schemas from different domains. For each domain, a pair of schemas developed independently was used for testing. We ensured that the CMs associated with the pair of schemas were also mutually independent. The independence was sought by using different domain ontologies 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 [1].

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	4	0.384
UTDB	13	CS dept. onto.	62		
HotelA	6	hotelA onto.	7	3	0.158
HotelB	5	hotelB onto.	7		
NetworkA	18	networkA onto.	28	5	0.106
NetworkB	19	networkB onto.	27		

Table 1: Characteristics of Test Data

The first three pairs of schemas were obtained from Clio’s test datasets. DBLP 1&2 are the relational schemas for the DBLP bibliography. They are associated with the Bibliographic 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 reversely engineered. Amalgam 1&2 are test

schemas developed in the Clio project. Fortunately, their original conceptual models are available. 3Sdb 1&2 are two versions of a repository of data on biological samples explored during gene expression analysis. They are shown in [8], which demonstrates how schemas evolve based on changing goals for the same application domain. Mappings would have to be discovered between the schemas when data are translated from one version to the next. 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<sup>5</sup>. 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 1, the test data have a variety of complexities.

**Methodology:** We compared the semantic approach, presented in this paper, with the chase technique illustrated in Example 1.1. That is, logical associations are assembled in each schema by chasing the referential integrity constraints, and mappings are derived from pairs of logical associations covering some correspondences. Before the experiments, we manually created non-trivial mappings between each tested pairs (a trivial mapping is from a single source table to a single target table.) These manually-created mappings are used to compare the mapping performance of the different methods. The comparison is focused on the intrinsic abilities of the methods. Since in its raw form, the chase approach generates maximal sets of columns that can be grouped by a join, we first applied a heuristic that removed any unnecessary joins — ones that did not introduce new attributes not covered by correspondences. (Apparently, the Clio tool uses such a heuristic.) Only afterwards were the proposed mappings offered by each technique compared for identity with the desired manually-created mapping.

**Measures:** We use *precision* and *recall* to measure the performance of the methods. For a given schema pair, let  $I$  be the number of correct mappings generated for a given set of correspondences. If a total of  $P$  mappings are generated by an algorithm, and there are  $R$  manually-created mappings for the given set of correspondences, the two measures are computed as:  $precision = I/P$  and  $recall = I/R$ . For each domain, we compute the average precision and average recall over all tested mapping cases.

**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 1 shows that it took less than one second. This is comparable with the chase approach technique, which also took less than one second for mapping generation in our experiments.

<sup>5</sup><http://www.atl.external.lmco.com/projects/ontology/i3con.html>

Next, in terms of the measures, Figure 7 compares the average precisions of semantic and chase for all the domains. Figure 8 compares the average recalls.

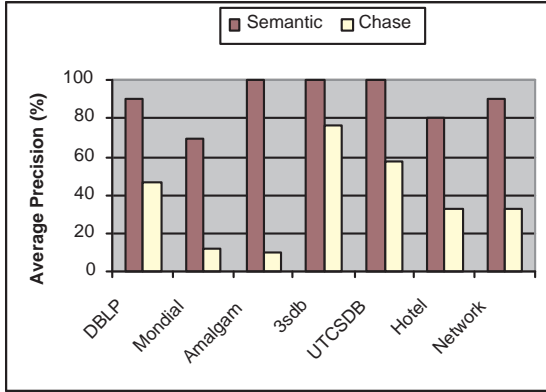


Figure 7: Average Precision

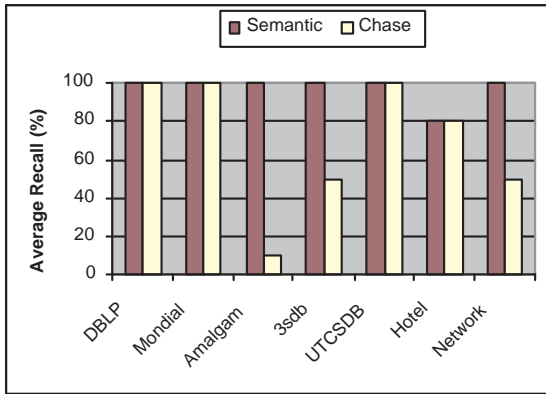


Figure 8: Average Recall

The results show that in general, the semantic approach performed at least as well as chase approach for the test datasets. The measures of recall show that in 6 out of 7 cases the semantic approach did not miss any correct mappings that were predicted by chase approach (since it got *all* the mappings sought), and made significant improvements in some cases. Moreover, Figure 7 shows that the semantic approach had significantly improved precision, at least by 20%. This is the major gain obtained by leveraging the semantics of schemas.

**Discussion:** We believe it is instructive to examine more closely the cause of the improvements. Our examination showed that the improvements were indeed made by using the richer structure of the CMs. For example, schema *Amalgam1* has a table: `Article(articleID, title, journal,...)` representing entity `Article`, while the schema *Amalgam2* has tables `allBibs(citKey)`, `citJournal(citKey,jrnIID)`, and `titles(citKey, title)`, and the foreign key constraints:

$$\begin{aligned} \text{citJournal.citKey} &\subseteq \text{allBibs.citKey}, \\ \text{titles.citKey} &\subseteq \text{allBibs.citKey}. \end{aligned}$$

The element correspondences specified were

`Article.title` ↔ `titles.title`, and,

`Article.journal` ↔ `citJournal.jrnIID`.

Note that the user did not link `Article.articleID` to `allBibs.citKey` at this point. The chase technique suggested the following two mappings:

$$M_1: \forall \text{articleID} \forall \text{title} \forall \text{journal} (\text{Article}(\text{articleID}, \text{title}, \text{journal} \dots) \rightarrow \exists x \text{allBibs}(x) \wedge \text{titles}(x, \text{title})).$$

$$M_2: \forall \text{articleID} \forall \text{title} \forall \text{journal} (\text{Article}(\text{articleID}, \text{title}, \text{journal} \dots) \rightarrow \exists y \text{allBibs}(y) \wedge \text{citJournal}(y, \text{jrnIID})).$$

Since both the `titles` entity and the `citJournal` entity are subclasses of the `allBibs` entity in the CM of *Amalgam2*, the semantic approach generated the following single mapping, which is the desired one:

$$M_3: \forall \text{articleID} \forall \text{title} \forall \text{journal} (\text{Article}(\text{inprocID}, \text{title}, \text{journal} \dots) \rightarrow \exists x \text{allBibs}(x) \wedge \text{citJournal}(x, \text{jrnIID}) \wedge \text{titles}(x, \text{title})).$$

The semantic approach discovered  $M_3$  directly by leveraging the semantics of the *Amalgam2* schema.

## 6 Related Work

The most directly related work is obviously *Clio* [13, 14], and we have already provided some comparisons of the basic techniques. The work presented here can be thought of as increasing recall by slightly generalizing the RIC chase to repeatedly merging functional relationships onto the entities in the CM, where subclass is also treated as a functional relationship. And it increases precision by eliminating certain candidate logical relations which (i) are dominated by larger ones, (ii) cannot be consistently satisfied because of disjointness constraints, or (iii) mappings that pair relationships with suspiciously different semantics (many-to-many with many-to-one, **partOf** with **non-partOf**).

Schema matching [10, 15] is the problem of identifying semantic relations between schema elements based on their names, data types, constraints, and schema structures. The primary goal is to find the one-to-one simple correspondences which are part of the input for building declarative schema mappings. Realizing that evidences carried by schemas themselves are not sufficient to derive more accurate matching results, some work [9, 17] utilize a corpus of schemas or a domain ontology in matching discovery. Their works are complementary to ours.

Conceptual models have been used in developing graphical query interfaces for databases. A central problem is the query inference when a user has marked nodes on a CM diagram. [18] applies the concept of maximal object from relational database theory to find a default connection among a set of nodes in a CM diagram. Assuming a user wants only one object to be used to infer a meaningful con-

nection, [16] uses the “minimum cost” object for the connection after assigning unit weights to functional edges and sufficiently large weights to non-functional edges to penalize them. (Incidentally, one of features in our approach can be viewed as a combination of the advantages of the above two approaches.) However, the fundamental difference is that we aim at finding a pair of matched connections in different CM graphs.

Last but not least, various algorithms for answering queries using views [7] shed insightful light on our problem for translating a tree over a CM graph into an algebraic expression in the logical schema.

## 7 Conclusions

Relational schema mapping is a problem that has been receiving considerable interest recently. We have proposed here an approach to discovering such relational schema mappings from simple table column correspondences, which leverages the semantics of the tables, expressed through connections to conceptual models. We first showed several cases where the current solution to discovering mapping from correspondences (based on “chasing” referential integrity and key constraints) does not produce the best results. We then developed algorithms for discovering plausible mappings at the conceptual level, and translated them into sketches of relational level mappings. Experimental results demonstrated that the semantic approach achieved a better performance on the test datasets drawn from a variety of domains.

Given the additional significant work that has gone into the Clio tool, it is 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 our own earlier tools, then the present technique could be used inside Clio to achieve better results in those cases where Clio’s initial mappings are judged to be weak at a first glance.

One direction for future work involves representing and using richer semantics than those represented by s-trees. In particular, we know that semantics presented as conjunctive queries over the CM can be represented as s-trees with additional equality edges, and comparisons of attributes with constants is also easy to add. Capturing the semantics of negated entities would be useful in representing so-called “horizontal partitioning” (e.g., table Employee only holds non-engineers and non-programmers). More general semantic definitions, as expressed by description logic concepts may also be useful. In these cases, the problem of rewriting the graphs into expressions over tables becomes harder, and may need to be solved knowing that the graphs were constructed in a special way using subgraphs representing table semantics. Additional work on ranking alter-

native mappings based on CMs would also be interesting. We also plan to investigate the related problem of finding complex semantic mappings between two CMs/ontologies, given a set of element correspondences.

## References

- [1] <http://www.cs.toronto.edu/~yuana/research/maponto/schemaMapping>.
- [2] Y. An, A. Borgida, and J. Mylopoulos. Constructing Complex Semantic Mappings between XML Data and Ontologies. In *ISWC’05*, 2005.
- [3] Y. An, A. Borgida, and J. Mylopoulos. Inferring Complex Semantic Mappings between Relational Tables and Ontologies from Simple Correspondences. In *ODBASE’05*, 2005.
- [4] Y. An, A. Borgida, and J. Mylopoulos. Discovering the Semantics of Relational Tables through Mappings. *Journal of Data Semantics*, VII, 2006.
- [5] A. Borgida and J. Mylopoulos. Data Semantics Revisited. In *SWDB’04 in Conjunction with VLDB’04*, pages 9–26, August 2004.
- [6] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering Complex Semantic Matches between Database Schemas. In *SIGMOD’04*, pages 383–394, 2004.
- [7] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal: Very Large Data Bases*, 10(4):270–294, 2001.
- [8] 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.
- [9] J. Madhavan, P. Bernstein, A. Doan, and A. Halevy. Corpus-Based Schema Matching. In *ICDE’05*, pages 57–68, 2005.
- [10] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *VLDB’01*, pages 49–58, 2001.
- [11] V. M. Markowitz and A. Shoshani. Representing Extended Entity-Relationship Structures in Relational Databases: A Modular Approach. *ACM TODS*, 17(3), September 1992.
- [12] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: a versatile graph matching algorithm and its application to schema matching. In *18th ICDE*, 2002.
- [13] R. J. Miller, L. M. Haas, and M. A. Hernandez. Schema Mapping as Query Discovery. In *VLDB’00*, pages 77–88, 2000.
- [14] L. Popa, Y. Velegrakis, R. J. Miller, M. Hernandez, and R. Fagin. Translating Web Data. In *VLDB’02*, pages 598–609, 2002.
- [15] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10:334–350, 2001.
- [16] J. A. Wald and P. G. Sorenson. Resolving the Query Inference Problem Using Steiner Trees. *ACM TODS*, 9(3):348–368, 1984.
- [17] 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.
- [18] Z. Zhang and A. O. Mendelzon. A Graphical Query Language for Entity-Relationship Databases. In *ER’83*, pages 441–444, 1983.