Finding Semantic Mappings from Relational Tables to Conceptual Models/Ontologies

Yuan An
Computer Science
University of Toronto
Canada
yuana@cs.toronto.edu

Alex Borgida
Computer Science
Rutgers University
USA
borgida@cs.rutgers.edu

John Mylopoulos
Computer Science
University of Toronto
Canada
jm@cs.toronto.edu

Abstract

Data integration is one of many problems requiring a semantic account of a database schema. At its best, such an account consists of a formal mapping between the schema and a formal conceptual model or ontology (CM) of the domain. This paper describes the underlying principles, algorithms, and a prototype of a tool which proposes such semantic mappings when given as input a relational schema, a CM, and *simple correspondences* between table columns and attributes of concepts in the CM. Although the algorithm presented is necessarily heuristic, we offer formal results stating that the answers returned are "correct" for relational schemas designed according to standard Entity-Relationship techniques. We also report on experience in using the tool with public domain schemas and ontologies.

1 Introduction and Motivation

A number of important database problems have been shown to have improved solution when one has available a *representation of the precise semantics of the database schema*. These include view integration, federated databases, data warehousing [2], and especially information integration through mediated schemas [11]. (See survey [22].) Since much information on the web is generated from databases (the "deep web"), the recent call for a Semantic Web, which requires a connection between web content and ontologies, provides additional motivation for the problem of associating semantics with data (e.g., [7]). In almost all of these cases semantics of the data is captured by a conceptual model or ontology (CM) of the problem domain, and some kind of *semantic mapping* between the database schema and the CM. Although sometimes the mapping is just a simple association from terms to terms, in other cases [11, 2, 1] what is required is a complex formula, often expressed in logic or a query language.

So far, it has been assumed that *humans* provide both the CM, and the semantic mapping between the CM and database schema. Unfortunately, this involves two steps, both of which are time-consuming and error prone: (i) building the CM; (ii) expressing the connection of the database schema to the CM as a mapping formula. For the first problem, we suggest *reusing* formal CMs that have already been developed. This view is particularly appropriate for the semantic web, where large ontologies, such as Cyc [15], are developed independently of any particular application. The same is true for information integration/data warehousing, where new databases may want to "join" after a CM has already been developed for the domain of discourse. For the second problem, we propose to build tools that assist users in finding complex mapping formulas — the subject of this paper.

Setting: To be explicit, we assume a situation where we are given a relational schema consisting of a set of relational tables, and a formal, independently developed CM, which can be potentially large (e.g., thousands of concepts). Our tool has users who are familiar with the database schema whose semantic mapping is being sought (both its schema and contents), but who are unfamiliar with the details of the CM, and are uncomfortable with writing complex logical formulas defining mappings. Note that the CM is not assumed to be *exactly* about the same domain of discourse as the database: it could be much richer, as would be the case with large ontologies.

Approach: The scenario we imagine is interactive, and involves tool support for several phases: (1) finding and expressing *simple correspondences* between relational table columns and CM components (mostly string/integer-valued attributes), which supply data to be stored in those columns; (2) providing a list of candidate formulas (expressed as conjunctive queries) that express the mappings between the table and the CM; (3) if necessary, supporting the user in inspecting, pruning and possibly editing these candidate mappings to refine them.

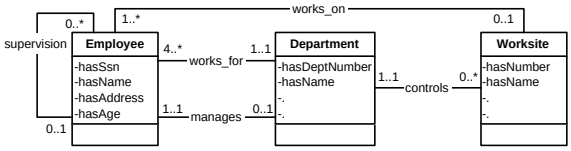

Figure 1: Company CM.

Our approach is directly inspired by the Clio project [14, 17], which developed a successful tool that infers mappings from one set of relational tables and/or XML documents to another, given just a set of correspondences between their respective attributes. As in [17], we focus here on step (2) of the process described above, since there is considerable existing work concerning the first step, in the area of schema matching [19]. Additional evidence for the importance of supporting steps other than just (1) is provided by the results of a survey concerning the distribution of time/effort in solving data integration problems [21]. The following example illustrates the input/output behavior of the tool we seek:

Example 1.1 *Given the CM in Figure 1 (expressed here in UML notation), relational table* Emp(*ssn*, name, dept, proj) *with key ssn, and the (natural) correspondences*
$\mathcal{T}$: *Emp.ssn*$\leftrightsquigarrow\mathcal{O}$: *Employee.hasSsn*
$\mathcal{T}$: *Emp.name*$\leftrightsquigarrow\mathcal{O}$: *Employee.hasName*
$\mathcal{T}$: *Emp.dept*$\leftrightsquigarrow\mathcal{O}$: *Department.hasDeptNumber*
$\mathcal{T}$: *Emp.proj*$\leftrightsquigarrow\mathcal{O}$: *Worksite.hasNumber*
we may expect a mapping of the form

$\mathcal{T}$*:Emp(ssn, name, dept, proj):-*
$\mathcal{O}$*:Employee*(x_1), $\mathcal{O}$*:hasSsn*(x_1,*ssn*),
$\mathcal{O}$*:hasName*(x_1,*name*), $\mathcal{O}$*:Department*(x_2),
$\mathcal{O}$*:works_for*(x_1,x_2), $\mathcal{O}$*:hasDeptNumber*(x_2,*dept*),
$\mathcal{O}$*:Worksite*(x_3), $\mathcal{O}$*:works_on*(x_1,x_3),
$\mathcal{O}$*:hasNumber*(x_3,*proj*).

where, for clarity, we used here prefixes $\mathcal{T}$ *and* $\mathcal{O}$ *to distinguish predicates in the relational schema and the CM.* □

Note that, as mentioned, the problem is inherently under-specified: in the example above, the table could just as well have represented the *manages* relationship, instead of *works_for*, so we currently propose to return all such semantic mappings. Without going into further details at this point, we summarize the contributions which we feel are being made here:

- The paper identifies a new version of the data mapping problem: that of *discovering* formulas expressing the semantic mapping between relational database schemas and formal CMs, encoded as graphs. (Exactly such mapping formulas are used, for example, in the DWQ system [2].)
- It then proposes a (necessarily heuristic) solution to this problem, whose most basic heuristic is a version of Occam's razor: finding something close to a minimal connection between the concepts that have attributes corresponding to table columns. Formally, these connections are so-called Steiner trees.
- The algorithm is enhanced to take into account information about the schema (key and foreign key structure), the ontology (cardinality restrictions), and standard database schema design guidelines.
- To gain empirical confidence, the proposed algorithms have been implemented, and evaluated in a number of experiments.
- To gain theoretical confidence, we describe formal results which state that if the schema was designed from a CM using techniques well-known in the Entity Relationship literature (which provide a natural semantic mapping for each table), then the tool will report essentially all and only the appropriate semantics. This shows that our heuristics are not just shots in the dark: in the case when the CM has no extraneous material, and when a table's schema has not been denormalized, the algorithm will produce reasonable results.

The rest of the paper is structured as follows. Section 2 discusses related work, and Section 3 presents the necessary background and notation. Section 4 describes an intuitive progression of ideas underlying our approach, while Section 5 provides the mapping discovery algorithm. In Section 6 we report on the prototype implementation of these ideas and experience with the prototype. Finally, Section 7 summarizes the contributions of this work and discusses directions for future research.

2 Related Work

The problem of discovering semantic connections dates as far back as Ross Quillian's PhD thesis [18], where a program, given noun-noun pairs, finds semantic paths between them within a semantic network. Such paths were used to generate English descriptions of the meaning for each pair.

The problem of *data reverse engineering* is to extract an ER diagram, for example, from a database schema. Sophisticated algorithms and approaches to this have appeared in the literature over the years (e.g., [12, 6]). The major difference between data reverse engineering and our work is that we are given an existing CM, and want to interpret a legacy relational schema in terms of it, whereas data reverse engineering aims to construct a new CM.

Schema matching (e.g., [19]) identifies semantic relations between schema elements based on their names, data types, constraints, and schema structures. Recent work on iMAP [4], discovers not only 1-1 matches between pairs of elements, but also *complex matches*

that specify how some combinations (e.g., concatenation) of elements correspond in the two schemas. iMAP concentrates on finding sophisticated formulas involving mostly single tables, though its multi-modal algorithm examines such hints as stored queries and extensional data to also discover join paths between tables. The present work can be thought of as focusing exclusively on finding such join paths, but based on table structure and semantic information in the CM, since neither queries nor instances are normally available for ontologies.

Relationship to Clio. As mentioned earlier, the Clio tool [14, 17] discovers formal queries describing how target schemas can be populated with data from source schemas. The present work could be viewed as extending this to the case when the source schema is a relational database, while the target is a CM. If one viewed the conceptual model as a relational schema made of unary and binary tables (e.g., $\mathcal{O}$*:Employee(X)*, $\mathcal{O}$*:hasSsn(X,ssn)*), one could in fact try to apply directly the Clio algorithm to Example 1.1, pushing it beyond its intended application domain. The desired mapping formula from Example 1.1 would not be produced for several reasons: (i) Clio [17] does not make a so-called logical table connecting $\mathcal{O}$*:hasSsn* and $\mathcal{O}$*:hasDeptNumber*, since the chase algorithm only follows foreign key references *out* of tables. (ii) The fact that *ssn* is a key by itself, leads us to prefer (see Section 4) a many-to-one relationship, such as *works_for*, over some many-to-many relationship $\mathcal{O}$*:previouslyWorkedFor*, which could have been part of the CM; 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 *unreasonable* options that arise in mapping to CMs.

3 Formal Preliminaries

We assume the reader is familiar with standard notions of relational databases, as presented in [20] for example. We will use the notation $T[\underline{K}, Y]$ to represent a relational table T with columns KY, and key K. (When we view tables as predicates, we will sort the arguments in alphabetical order.) If necessary, we will refer to the individual columns in Y using $Y[1], Y[2], \ldots$, and use XY as concatenation. Our notational convention 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. (Note that we use the terms "table" and "column" when talking about relational schemas, reserving "relation(ship)" and "attribute" for aspects of the CM.) A foreign key (fk) in T is a set of columns F that *references* table T', and imposes a constraint that the projection of T on F is a subset of the projection of T' on key(T').

We use a generic conceptual modeling language (CML), which contains *common* aspects of most semantic data models, UML, ontology languages such as OWL [16], and description logics. Specifically, the language allows the representation of *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 paper. Concepts are organized in the familiar **is-a** hierarchy. Object properties, and their inverses (which are always present), are subject to constraints such as specification of domain and range, plus the familiar 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 directed *ontology graph*, which has concept nodes labeled with concept names C, and edges labeled with object properties p; for each such p, there is an edge for the inverse relationship, referred to as p^-. For each attribute f of concept C, we create a separate attribute node $N_{f,C}$, whose label is f, and with edge labeled f from C to $N_{f,C}$.[1] In figures, we follow UML, drawing concepts as rectangles, with attributes listed inside.

We propose to have edge p from C to B, written in the text as `C ---p--- B`, to represent that p has domain C and range B. (If the relationship p is functional, we write `C ---p->-- B`.) For expressive CMLs such as OWL, we may also connect C to B by p if we find an existential restriction stating that each instance of C is related to *some* instance of B by p. The significant feature of the resulting graph is that if C and B (or their super-classes) are **not** linked by edge p, then a 'join' formula $A(x), p(x, y), B(y)$ will **not** be satisfied for any x, y, and hence should not appear in any semantic mapping.

In this paper, a *correspondence* $T.c \rightsquigarrow E.f$ will relate column c of table T to attribute f of concept E. Since our algorithms deal with ontology graphs, formally a correspondence L will be a mathematical relation $L(T, c, E, f, N_{f,E})$, where the first two arguments determine unique values for the last three. Given L, T, and c, we will frequently want to find E; for this, we use the derived function $\mathsf{onc}_L(c, T)$, and since the context will always identify L and T, we will use the expression onc(c). We extend onc to sets of columns in a point-wise manner. Therefore the set M=onc(columns($T1$)) is the set of entities in the CM such that each has some attributes corresponding to a column of $T1$ according to the correspondences implicit in the context. Finally, a *semantic mapping* has the form of a Horn-clause $T(X) : -\psi(X, Y)$, where T is a table with columns X (which become arguments

[1] Unless ambiguity arises, we will use "node C", when we mean "concept node labeled C".

to its predicate), and ψ is a conjunctive formula over predicates representing the CM, with Y existentially quantified as usual.

Using the terminology of [11, 10], and following the example of [11, 2], we are looking for a Local-as-View (LAV) mapping connecting the schema with the CM, partly because our tool is particularly well suited to the incremental connection of databases to conceptual models, which is associated with the LAV approach. In the future, we plan to also explore the discovery of GAV and GLAV mappings.

4 Principles of Mapping Discovery

As mentioned in Section 2, Quillian [18] was searching for semantic connections between word senses/nodes in a semantic network. His solution relied on finding *shortest paths* between them. Similarly, we are starting from the concepts/nodes in the set $M = \{\mathsf{onc}(c)|c \in \mathsf{columns}(T)\}$, and are trying to discover semantic connections between them, so it makes sense to look for "shortest" connections between the nodes in M. A *spanning tree* S_M for M in the graph of the CM captures this notion, in the sense that it uses the minimal number of edges necessary to connect M, and does not introduce any extraneous concepts.

Thus, for Example 1.1, we would get S^1_{emp} as one spanning tree, where S^1_{emp} has edges `Employee ---works_for-->- Department`, `Employee ---works_on-->- Worksite`, plus edges for the attributes in the correspondence, such as `Employee ---hasSsn-->-` $\mathtt{N}_{hasSsn,E}$, where $\mathtt{N}_{hasSsn,E}$ is an attribute node labeled with `hasSsn`.

A spanning tree S for table $T[Y]$, based on correspondence L, gives rise to a conjunctive formula according to the procedure encodeTree(S,L) below, which basically assigns variables to nodes, and connects them using edge labels as predicates:

Function encodeTree(S,L)
input: subtree S of ontology 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:

1. Suppose N is the root of S. Let $\Psi = \{\}$.
2. if N is an attribute node with label f,
   find d such that $L(\_, d, \_, f, N) = true$, return $(d, true)$. */*for leafs use corresp. column names*/*
3. 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
   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;
   return (x, Ψ).

Continuing with the example above, suppose encodeTree(S^1_{emp},L) assigns variables x_1, x_2, and x_3 to *Employee*, *Department*, and *Worksite*, respectively; it will then generate the right-hand side of the Horn-clause in Example 1.1, where, for example, $works\_for(x_1, x_2)$ represents the edge `Employee ---works_for-->- Department`, while $hasSsn(x_1, ssn)$ represents the edge `Employee ---hasSsn-->-` $\mathtt{N}_{hasSsn,E}$, because of the correspondence $Emp.ssn \leftrightsquigarrow Employee.hasSsn$. As noted before, there are alternate semantics, based on different spanning trees, which often arise due to multiple edges between concepts.

To reduce the complexity of the algorithms, and the size of the answer set, we modify the graph by collapsing multiple edges between nodes E and F, labeled $p_1, p_2, \ldots$ say, into a single edge labeled $'p_1; p_2; \ldots'$ The idea is that it will be up to the user to choose between the alternative labels in phase (3) of the scenario described in the Section 1, 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.

Consider however the case when $T[\underline{c}, b]$ is a table with key c, corresponding to attribute c on C, and b is a foreign key corresponding to b on B, where b and c are unique identifier attributes within their classes. 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. Trees for such formulas look like functional edges, and hence should be preferred. For example, given table $Dep[\underline{dept}, ssn, \ldots]$, and correspondences like in Example 1.1, the proper spanning tree uses $manages^-$ rather than $works\_for^-$.

Conversely, for table $T'[\underline{c, b}]$, 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 will work in two stages: first connecting the concepts corresponding to key columns into a *skeleton tree*, then connecting the remaining columns to the skeleton by, preferably, functional edges.

Finally, we must deal with our assumption that the CM is developed independently, which implies that not all parts of it are reflected in the database schema. This complicates things, since in building the spanning tree we may need to go through additional nodes, which end up not being present in the database. For example, consider the table *Project (<u>name</u>, supervisor, ...)*, with *name* as key, and correspondences
$\mathcal{T} : Project.name \leftrightsquigarrow \mathcal{O} : Worksite.hasName$, and
$\mathcal{T} : Project.supervisor \leftrightsquigarrow \mathcal{O} : Employee.hasSsn$,
to the CM in Figure 1. Based on the argument above, the edge $works\_on^-$, connecting *Worksite* to *Employee* is inappropriate because it is not functional. Instead,

we prefer the *functional path* `controls`$^-$`.manages`$^-$, passing through node *Department*, even if it is longer.

We therefore modify our algorithm to look for so-called *Steiner-trees* [2] (s-trees for short), instead of ordinary spanning trees. (This is, of course, modulo other considerations such as key and functional paths.) 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 topic widely covered in even undergraduate database courses [20]. (We call this *er2rel schema design*.) One benefit of this approach will be to allow 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 are realized in the database. We emphasize that our algorithm uses the general principles enunciated above even in such cases, with relatively good results in practice.

5 Mapping Discovery Algorithms

We will introduce the details of the algorithm in a gradual manner, by repeatedly 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 [20], though we summarize the ideas.

5.1 An Initial Subset of ER notions

We start with a subset of ER that contains the notions such as entity set (called just "entity" here), with attributes, and binary relationship set. 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 in our CM. A binary relationship set corresponds to two relationships in our CM, one for each direction, though only one is mapped to a table. Such a relationship will be called *many-many* if neither it nor its inverse is functional. A *strong entity* $\mathbb{S}$ has some attributes that act as identifier. We shall refer to these using $\mathsf{unique}(\mathbb{S})$ when describing the rules of schema design. A *weak entity* $\mathbb{W}$ has instead $\mathsf{localUnique}(\mathbb{W})$ attributes, plus a functional total binary relationship p (preferred to with $\mathsf{idRel}(\mathbb{W})$) to an identifying owner entity (preferred to with $\mathsf{idOwn}(\mathbb{W})$).

Note that information about general identification cannot be represented in even highly expressive 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 recovery. The er2rel design methodology (we follow mostly [12, 20]) is defined by two components: To begin with, Table 1 specifies a mapping $\tau(O)$ returning a relational table schema for every CM component O, where O is either a concept/entity or a binary relationship. In this subsection, we assume that no pair of concepts is related by more than one relationship, and that there are no so-called "recursive" relationships relating an entity to itself. (We deal with these in Section 5.3.)

ER Model object O	**Relational Table τ(O)**
Strong Entity $\mathbb{S}$	*columns:* X
	primary key: K
Let X=attribs($\mathbb{S}$)	*fk's:* none
Let K=unique($\mathbb{S}$)	*anchor:* $\mathbb{S}$
	semantics:
	$T(X)$:-$\mathbb{S}(y)$,hasAttribs(y, X).
	identifier:
	identify$_{\mathbb{S}}(y, K)$:-$\mathbb{S}(y)$,hasAttribs(y, K).
Weak Entity $\mathbb{W}$	*columns:* ZX
let	*primary key:* UX
Z=attribs($\mathbb{W}$)	*fk's:* X
X = key(τ(idOwn($\mathbb{W}$)))	*anchor:* $\mathbb{W}$
U =localUnique($\mathbb{W}$)	*semantics:*
$\mathbb{E}$ = idOwn($\mathbb{W}$)	$T(X, U, V)$:-
$V = Z - U$	$\mathbb{W}(y)$, hasAttribs(y, Z),$\mathbb{E}(w)$,
	idrel($\mathbb{W}$)(y, w),identify$_{\mathbb{E}}(w, X)$.
	identifier:
	identify$_{\mathbb{W}}(y, XU)$:-
	$\mathbb{W}(y)$,$\mathbb{E}(w)$,idrel($\mathbb{W}$)(y, w),
	hasAttribs(y, U),identify$_{\mathbb{E}}(w, X)$.
Functional Relationship $\mathbb{F}$	*columns:* X_1X_2
$\mathbb{E}_1$–$\mathbb{F}$->-$\mathbb{E}_2$	*primary key:* X_1
	fk's: X_i references $\tau(\mathbb{E}_i)$,
let X_i = key($\tau(\mathbb{E}_i)$)	*anchor:* $\mathbb{E}_1$
for $i = 1, 2$	*semantics:*
	$T(X_1, X_2)$:-
	$\mathbb{E}_1(y_1)$,identify$_{\mathbb{E}_1}(y_1, X_1)$, $\mathbb{F}(y_1, y_2)$,
	$\mathbb{E}_2(y_2)$, identify$_{\mathbb{E}_2}(y2, X_2)$.
Many-many	*columns:* X_1X_2
Relationship $\mathbb{M}$	*primary key:* X_1X_2
$\mathbb{E}_1$–$\mathbb{M}$–$\mathbb{E}_2$	*fk's:* X_i references $\tau(\mathbb{E}_i)$,
let X_i = key($\tau(\mathbb{E}_i)$)	
for $i = 1, 2$	*semantics:*
	$T(X_1, X_2)$:-
	$\mathbb{E}_1(y_1)$,identify$_{\mathbb{E}_1}(y_1, X_1)$, $\mathbb{M}(y_1, y_2)$,
	$\mathbb{E}_2(y_2)$, identify$_{\mathbb{E}_2}(y2, X_2)$.

Table 1: er2rel Design Mapping.

[2] A Steiner-tree for set M of nodes in graph G is a minimum-weight subgraph of G that contains all the nodes in M and is a tree. In the case when M is large, computing an s-trees may become too expensive since the problem is NP-hard. However, we will only compute s-trees for the concepts corresponding to columns in keys, and key cardinality is small in practice.

In addition to the schema (columns, key, fk's), Table 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 spanning tree);
- a formula for the semantic mapping for the table, expressed as a Horn rule with head $T(V)$ (this is what our algorithm should be recovering); in the body of the Horn formula, the function $\mathsf{hasAttribs}(x, Y)$ returns conjuncts $L_0(Y[j])(x, Y[j])$ for the individual columns Y[1], Y[2],... in Y, where, for the purposes of exposition, L_0 describes a trivial correspondence: the identity mapping from attribute names to table columns.
- the formula for a predicate $\mathsf{identify}_C(x, Y)$, showing how object x in (strong or weak) entity class C can be identified by values in Y[3].

Note that τ is defined recursively, and will only terminate if there are no "cycles" in the CM (see [12] for definition of cycles in ER).

The er2rel methodology also suggests that the schema generated using 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 $T_0(K, V) : -\phi(K, V)$, and of T_1 is $T_1(K, W) : -\psi(K, W)$, then the semantics of table T=$\mathsf{merge}(T_0,T_1)$ is, to a first approximation, $T(K, V, W) : -\phi(K, V), \psi(K, W)$. And the anchor of T is the entity E.

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, say).

Now we turn to the algorithm for finding the semantic connections between nodes in the set M=$\{\mathsf{onc}(c)|c \in \mathsf{columns}(T)\}$. As mentioned in the previous section, because the keys of a table functionally determine the rest of the columns, the algorithm for finding the semantic connections 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.
3. Link any unaccounted-for attributes by arbitrary shortest paths to the tree.

More specifically, the main function, $\mathsf{getTree}(T,L)$, will find the semantics of table T, given correspondence L, by returning an s-tree S, whose translation using $\mathsf{encodeTree}(S,L)$ yields the conjunctive formula defining the semantics of table T.

The function $\mathsf{getTree}(T, L)$ makes calls to function $\mathsf{getSkeleton}$ on T and other tables referenced by fks in T, in order to get a set of (skeleton tree, anchor)-pairs, which have the property that in the case of er2rel designs, if the anchor returned is concept C, then the encoding of the s-tree is the formula for $\mathsf{identify}_C$. Finally, $\mathsf{getTree}(T,L)$ connects all nodes using meaningful paths.

Function $\mathsf{getSkeleton}(T,L)$
input: table T, correspondences L for $\mathsf{key}(T)$
output: a set of (s-tree, anchor) pairs
steps:
Suppose $\mathsf{key}(T)$ contains fks $F_1,\ldots,F_n$ referencing tables $T_1(K_1),..,T_n(K_n)$;

1. If $n \leq 1$ and $\mathsf{onc}(\mathsf{key}(T))$ is just a singleton set $\{C\}$, then return $(C, \{C\})$.[4] /*Likely a strong entity: the base case.*/
2. Else, let L_i=$\{T_i.K_i \rightsquigarrow L(T, F_i)\}$/*translate corresp's thru fk reference*/; compute $(Ss_i, Anc_i) = \mathsf{getSkeleton}(T_i, L_i)$.
   (a) If $\mathsf{key}(T) = F_1$, then return (Ss_1, Anc_1).
   (b) If $\mathsf{key}(T)$=F_1A, where columns A are not in any foreign key of T then /*possibly a weak entity*/
      i. if $Anc_1 = \{N_1\}$ and $\mathsf{onc}(A) = \{N\}$ such that there is a total functional path π from N to N_1, then return $(\mathsf{combine}(\pi, Ss_1), \{N\})$.[5]
      ii. else let $Ns = Anc_1 \cup \mathsf{onc}(A)$, and find s-tree Ss' connecting nodes in Ns; return $(\mathsf{combine}(Ss', Ss_1), Ns)$.
   (c) Else supposing $\mathsf{key}(T)$ has additional non-fk columns $A[1], \ldots A[m]$, $(m \geq 0)$; let Ns=$\{Anc_i\} \cup \{\mathsf{onc}(A[j]), j = 1, .., m\}$, and find s-tree Ss' connecting the nodes in Ns, where any pair of nodes in Ns is connected by a many-many path;
   return $(\mathsf{combine}(Ss', \{Ss_j\}), Ns)$.

In order for $\mathsf{getSkeleton}$ to terminate, it is necessary that there be no cycles in fk references in the schema. Such cycles (which may have been added to represent additional integrity constraints, such as the the fact that an association is total) can be eliminated from a schema by replacing the tables involved with their outer join over the key. $\mathsf{getSkeleton}$ deals with strong entities and their functional relationships in step (1), with weak entities in step (2.b.i), 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

[3]This is needed in addition to $\mathsf{hasAttribs}$, because weak entities have identifying values spread over several concepts.

[4]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.

[5]Function $\mathsf{combine}$ merges edges of trees into a larger tree.

them with paths that are not functional, even when every edge is reversed. The interesting property of getSkeleton is that if $T = \tau(C)$ according to the er2rel rules in Table 1, where C corresponds to a (strong or weak) entity, then getSkeleton returns (S, Anc), where $Anc = C$ as anchor, and encodeTree(S) is logically equivalent to identify$_C$. Similarly, if $T = \tau(p)$, where p is a functional relationship originating from concept C, in which case its key looks just like an entity key.

Turning to getTree, we have

Function getTree(T,L)
input: table T, correspondences L for columns(T)
output: set of s-trees [6]
steps:

1. Let L_k be the subset of L induced by key(T); compute (S', Anc')=getSkeleton(T, L_k).
2. If onc(nonkey(T)) − onc(key(T)) is empty, then return (S', Anc'). */*getSkeleton did all the work already*/*
3. For each foreign key F_i in nonkey(T) referencing $T_i(K_i)$: let $L^i_k = \{T_i.K_i \rightsquigarrow L(T, F_i)\}$, and compute (Ss''_i, Anc''_i) = getSkeleton(T_i, L^i_k).

   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 fk, let N = onc(c); find π=shortest functional path from Anc' to N; update S := combine(S, π).
5. In all cases above asking for functional paths, use a shortest path if a functional one does not exist.
6. Return S.

Having seen the general algorithm, we can now state its desirable properties. Since the precise statement of theorems (and algorithms) is quite lengthy and requires a lot of minute details for which we do not have room here, we express the results as "approximately phrased" propositions. First, getTree finds the desired semantic mapping, in the sense that

Proposition 5.1 *Let table T be part of a relational schema obtained by er2rel derivation from conceptual model $\mathcal{E}$. Then some tree S returned by* getTree(T) *has the property that* encodeTree(S, L_0) *is logically equivalent to the semantics assigned to T by the er2rel design.*

Note that this "completeness" result is non-trivial, since, as explained earlier, it would not be satisfied by the current Clio algorithm [17], 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

Proposition 5.2 *If S' is any tree returned by* getTree(T)*, with T as above, then* encodeTree(S', L_0) *represents the semantics of* some *table T' derivable by er2rel design from $\mathcal{E}$, where T' is isomorphic*[7] *to T.*

Such a result would not hold of an algorithm which returns only minimal spanning trees, for example.

We would like to point out that the above algorithm performs reasonably on some non-standard designs as well. For example, consider the table $T(\underline{personName}, cityName, countryName)$, where the column identifiers suggest natural correspondences to attributes of concepts $Person, City$ and $Country$ in a CM $\mathcal{E}$. If $\mathcal{E}$ 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:

T(personName,cityName,countryName):-
Person(x_1), City(x_2),Country(x_3),
bornIn(x_1,x_2),locatedIn(x_2,x_3),
pname(x_1,personName), cname(x_2,cityName),
crname(x_3,countryName).

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:

T(personName, cityName, countryName):-
Person(x_1), City(x_2), Country (x_3),
bornIn (x_1,x_2), citizenOf(x_1,x_3), ...

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.

5.2 Reified Relationships

It is highly desirable to have n-ary relationship sets connecting entities, and to allow relationship sets to have attributes (what are called "association classes" in UML). Unfortunately, these features are not directly supported in most CMLs, such as OWL, which only have binary relationships. Such notions must instead be represented by *"reified relationships"* [3]: 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 2. And a relationship attribute, such as when the buying occurred, becomes an attribute of the *Buys* concept, such as *whenBought*.

Unfortunately, reified relationships cannot be distinguished reliably from ordinary entities in normal CMLs on purely formal, syntactic grounds, yet they

[6] 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.

[7] Informally, two tables are isomorphic if there is a bijection between their columns which preserves key and foreign key structure.

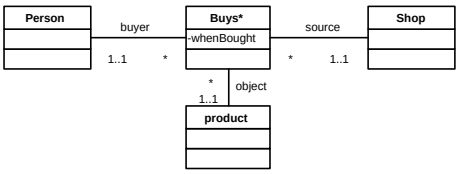

Figure 2: N-ary Relationship Reified.

need to be treated in special ways during recovery. For this reason we assume that they can be distinguished on *ontological grounds*. For example, in Dolce [5], they are subclasses of top-level concepts *Quality* and *Perdurant/Event*. For a reified relationship $\mathbb{R}$, we use functions $\mathsf{roles}(\mathbb{R})$ and $\mathsf{attribs}(\mathbb{R})$ to retrieve the appropriate (binary) properties.

The design τ of relational tables for reified relationships is shown in Table 2.

ER model object O	Relational Table $\tau(O)$
Reified Relationship $\mathbb{R}$	*columns:* $\mathrm{Z}X_1 \ldots X_n$
if $r_1, \ldots, r_n$ are roles of $\mathbb{R}$	*primary key:* $X_1 \ldots X_n$
let Z=$\mathsf{attribs}(\mathbb{R})$	*fk's:* $X_1, \ldots, X_n$
X_i=$\mathsf{key}(\tau(\mathbb{E}_i))$	*anchor:* $\mathbb{R}$
where $\mathbb{E}_i$ fills role r_i	*semantics:* $T(ZX_1 \ldots X_n)$:- $\mathbb{R}(y), r_i(y, w_i), \mathsf{hasAttribs}(y, Z), \mathbb{E}_i(w_i), \mathsf{identify}_{\mathbb{E}_i}(w_i, X_i), \ldots$
	identifier: $\mathsf{identify}_{\mathbb{R}}(y, \ldots U_i \ldots)$:- $\mathbb{R}(y), \ldots r_i(y, w_i), \mathbb{E}_i(w_i), \mathsf{identify}_{\mathbb{E}_i}(w_i, U_i), \ldots$

Table 2: er2rel Design for Reified Relationship.

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

- If $\mathsf{key}(T)$=$F_1F_2 \ldots F_n$ and there exist reified relationship $\mathbb{R}$ with n roles $r_1, \ldots, r_n$ pointing at the singleton nodes in $Anc_1, \ldots, Anc_n$ respectively, then let $S = \mathsf{combine}(\{r_j\}, \{Ss_j\})$, and return $(S, \mathbb{R})$.
- If there is no $\mathbb{R}$ exists, go to 2(c).

The main change to getTree is to 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. So after step 1. we add

- if $\mathsf{key}(T)$ is the concatenation of two foreign keys F_1F_2, nonkey(T) is empty, and a $\mathbb{R}$ has been found in step 1; 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 $(S') \cup (\mathsf{combine}(\rho, Ss_1, Ss_2))$

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

5.3 Replication

If we allow recursive relationships, or allow the merger of tables for different functional relationships connecting the same pair of concepts (e.g., *works_for* and *manages*), the mapping in Table 1 is incorrect because column names will be repeated in the multiple occurrences of the foreign keys. We will distinguish these (again, for ease of presentation) by adding superscripts as needed. For example, if *Person* is connected to itself by the *likes* property, then the table for *likes* will have schema $T[\underline{ssn^1}, \underline{ssn^2}]$. The correspondence L_0 implicit in τ is then extended the obvious way, to map c^k to c.

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 E. In such situations, the algorithm will first make a copy E_{copy} of node E in the ontology graph, as well as its attributes. E_{copy} participates in all the object relations E did, so edges must be added. After replication, we can set $\mathsf{onc}(c) = E$ and $\mathsf{onc}(d) = E_{copy}$, or $\mathsf{onc}(d) = E$ and $\mathsf{onc}(c) = E_{copy}$. 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 alternate semantics corresponding to *likes*, and its inverse, *likedBy*. (A different example would involve 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 both a functional edge `D --- p ->-- E` and its replicate `D --- p ->--` $\mathtt{E}_{copy}$, (or several replicates), since a function has a single value, and hence the different columns of a tuple will end up having identical values: a clearly poor schema.

5.4 Addressing Class Specialization

The ability to represent subclass hierarchies, such as the one in Figure 3 is a hallmark of CMLs and modern so-called Extended ER (EER) modeling.

Almost all textbooks (e.g., [20]) describe two techniques for designing relational schemas in the presence of class hierarchies

1. Map each concept into a separate table following the standard er2rel rules. This approach requires two adjustments: First, subclasses must inherit identifier 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 E' of entity E, its key $\mathsf{key}(\tau(E'))$ should also be set to reference as a foreign key $\tau(E)$, as a way of maintaining inclusion constraints dictated by the is-a relationship.

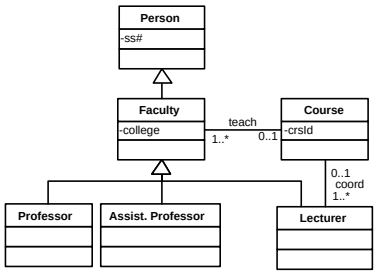

Figure 3: Specialization Hierarchy.

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

Some researchers also suggest a third possibility:

3. "Collapse up" the information about subclasses into the table for the superclass. This can be viewed as the result of $\mathsf{merge}(T_E, T_{E'})$, where $T_E[K, A]$ and $T_{E'}[K, B]$ are the tables generated for E and its subclass E' according to technique (1.) above. In order for this design to be "correct", [12] requires that $T_{E'}$ 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 E' can be distinguished from those of E).

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 $ss\#$ 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 ontology concepts are realized in the database schema, according to our scenario. For this reason, we have chosen to try to deal with some of the ambiguity in the first phase, during the establishment of correspondences. Specifically, the user is asked to provide a correspondence from column c to attribute f *on the lowest concept whose instances provide data appearing in the column.* Therefore, in the above example of table $T(\underline{ss\#}, crsId)$, $ss\#$ is made to correspond to *ssn* on *Faculty* in case (a), while in cases (b) and (c) it is made to 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 capture designs (1.) and (2.) above, we do not need to modify our earlier algorithm in any way, if 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 can add to the graph an actual edge for the inverse of the **is-a** relation: a functional edge labeled *alsoA*, with lower-bound 0: `E --- alsoA ->-- E`', connecting superclass E to each of its subclasses *E'*. It is then sufficient to allow functional paths between concepts to consist only of *alsoA* edges, in addition to the normal kind, in getTree.

5.5 Outer Joins

The observant reader has probably noticed that the definition of the semantic mapping for $T = \mathsf{merge}(T_E, T_p)$ was not quite correct: $T(\underline{K}, V, W) :- \phi(K, V), \psi(K, W)$ 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 $\lceil \mu(X, Y) \rceil^Y$, which will stand for the formula $\mu(X, Y) \vee (Y = null \wedge \neg \exists Z.\mu(X, Z))$, 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 $T(\underline{K}, V, W) :- \phi(K, V), \lceil \psi(K, W) \rceil^W$.

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.

6 Implementation and Experience

We have implemented the MAPONTO tool as a third-party plugin of the well-known knowledge-base development tool *Protégé* [9]. We conducted a series of experiments on publicly available database schemas and ontologies, which vary in size and complexity. Our goal was to evaluate the effectiveness and efficiency of the mapping algorithms on real world instances. We briefly summarize the results of our empirical experience with MAPONTO.

Schemas and Ontologies. Although obtaining existing real-world database schemas and CMs for our mapping experiments was a challenge, we managed to set up our experiments on relational schemas converted from many publicly available semi-structured data sources (in XML), and ontologies developed for the Semantic Web. Table 3 describes four pairs of database schemas and ontologies which we found. The #links in an ontology indicates the number of edges in the multi-graph. The university department schema describes the employee and student information of

the Department of Computer Science in University of Toronto; an academic department ontology is publicly available in the DAML library [8]. The academic conference database concerns VLDB, and an academic conference ontology was downloaded from the SchemaWeb ontology repository. The DBLP schema describes the well-known computer science bibliography; a bibliography ontology is available in the library of the Stanford's Ontolingua server. We also used MAPONTO to find semantic mappings from the 8 database tables related to library information which are posted on the web site of the Observer [13] project to the bibliography ontology above. We do not report on these individually since they are closely related and have overlaps. Finally, we selected a complex schema about countries from one of the database reverse engineering papers, and we used for it the CIA factbook ontology.

Domains	Schema		Ontology	
	# of tables	# of columns	# of concepts	# of links
University department	8	32	62	1913
Academic Conference	9	38	27	143
DBLP	5	27	75	1178
OBSERVER	8	115	75	1178
Country	6	18	52	125

Table 3: Schemas and ontologies for our experiments.

Experimental Setting and Summary of Results. MAPONTO was implemented in the Java language, and all experiments were run on a Dell desktop with 1.8GHZ Intel Pentium 4 CPU and 1G memory. A summary of the results, for a total of 28 relational tables, are listed in Tables 4 thru 7, which show the size of each relational table, the number of candidate semantic mappings generated, and the time for generating the candidate mappings. Notice that the number of candidates is the number of s-trees obtained by the algorithm. Also, a single edge of an s-tree may represent the multiple edges between two nodes, collapsed using our $p;q$ abbreviation. If there are m edges in a s-tree and each edge has n_i $i = 1, .., m$ original edges collapsed, then there are $\prod_i^m n_i$ original s-trees. (We show below a formula generated from such a collapsed s-tree.)

Mapping Accuracy. After loading the schemas and the ontologies, and executing the mapping algorithm, we found that MAPONTO generated single candidate semantics for 26 tables, and multiple candidates for an additional 2 tables. Among the 26 single responses, according to independent observation, 22 are the intended semantics. For the additional 8 relational tables from the OBSERVER project, (surprisingly many of which had "collapsed **is-a** hierarchies" — designs of type (3)), only one, mentioned below, was not assigned correct semantics.

Please note that since we collapsed multiple parallel edges into a single edge, some singleton candidates

Table Name	# of Cols	# of Candidates generated	Execution time(ms)
Student	6	1	113
AcademicStaff	5	1	153
AdminStaff	5	1	12
TechnicalStaff	4	1	12
AreaOfInterest	2	1	81
Roles	2	1	25
Course	6	10	145
TaAssignment	2	1	112

Table 4: Summary of Department Schema Results.

Table Name	# of Cols	# of Candidates generated	Execution time(ms)
Conference	4	1	8
Paper	6	1	8
PaperAuthor	2	1	18
Person	4	1	6
Registration	4	1	7
Workshop-Registration	2	1	14
Event	7	1	9
Chair	4	1	97
Presentation	5	1	26

Table 5: Summary of Conference Schema Results.

Table Name	# of Cols	# of Candidates generated	Execution time(ms)
DblpDoc	19	4	90
DblpAuthor	2	1	16
DblpPublisher	2	1	7
DblpCites	2	1	88
DblpWrites	2	1	40

Table 6: Summary of DBLP Schema Results.

Table Name	# of Cols	# of Candidates generated	Execution time(ms)
Country	4	1	160
CurrencyValue	3	1	156
CityPopulation	3	1	29
Export	3	1	77
Company	3	1	4
European	2	1	61

Table 7: Summary of Country Schema Results.

returned are shorthand for several Horn formulas, representing alternative semantics. Thus the following is one of the formulas generated by MAPONTO:

TaAssignment(courseName, studentName):-
Course(x_1), GraduateStudent(x_2),
[hasTAs;takenBy](x_1,x_2),
workTitle(x_1,courseName),
personName(x_2,studentName).

As suggested earlier, heuristics involving the names of tables/columns might help the user select the appropriate one in the final, post-processing phase.

It is instructive to consider various categories of problematic schemas/mappings, and the kind of future work they suggest.

(i) *Absence of tables expected by* er2rel. For example, we expect the connection *Person — researchInterest — Research* to be returned for the table $AreaOfInterest[\underline{name}, \underline{area}]$. 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. We are currently studying how to deal with the elimination, during schema design, of tables that represent finite enumerations, or can be recovered by projection from tables representing total many-many relationships

(ii) *Differences in Representation.* The table *Presentation (event, presenter, paper, start, end)* represents an n-ary relationship among three entities. However, in the ontology, the Presentation concept is a subclass of Event concept and has Presenter and Paper as role fillers.

(iii) *Mapping formula requiring select.* 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 ontology values. A similar example appears in [14].

(iv) *Non-standard design.* One of the bibliography tables had columns for *author* and *otherAuthors* for each document. MAPONTO found a formula that was close to the desired one, with conjuncts *hasAuthor(d, author)*, *hasAuthor(d, otherAuthors)*, but, unsurprisingly, could not add the requirement that *otherAuthors* is really the concatenation of all but the first author.

(v) *Complex multiple paths* The one remaining example in which multiple candidates were generated involved table *DblpDoc*, and resulted from multiple paths found from *Document* to the *Month-Name* concept. Since *Time-Point* has three subclasses each of which has an object property pointing to *Month-Name*, there are four different paths from the *Document* to the *Month-Name*. (We were pleasantly surprised by the small number of such occurrences, since an original implementation of MAPONTO, which only used s-tree heuristics, was more prolix.)

Efficiency. Tables 4- 7 indicate that execution times were not significant, since, as predicted, the search for Steiner trees and paths took place in a relatively small neighborhood.

7 Conclusion and Future Work

The problem of discovering semantic mappings between relational database schemas and formal CMs/ontologies is well-motivated by information integration applications [22, 2, 11]. We have proposed a solution to this problem, relying on information from the database schema (key and foreign key structure) as well as the CM (cardinality restrictions, **is-a** hierarchies). To gain empirical confidence, the proposed algorithms and heuristics have been implemented into a tool, and evaluated in a number of experiments. Moreover, to gain theoretical confidence, we have provided theorems which state that if a table's schema follows ER design principles then the tool will report essentially all and only the semantics implied by the ER-to-relational design. We emphasize however that MAPONTO works correctly on a much broader range of schemas, as shown by our experiments (where the schemas and CMs were developed independently) and as illustrated by the $Person[ssn, city, country]$ example in Section 5.1, and the $Project[\underline{pname}, supervisor, \ldots]$ example in Section 4. (The latter illustrates the success of generalizing functional edges to functional paths.) Moreover, this assurance shows that human users of the tool will be able to focus their effort on the minority of tables whose schema has been "denormalized" from the original ideal design.

Our work extends the pioneering work on the Clio project in scope (handling conceptual schemas), mapping discovery techniques (based on features of conceptual schemas, and key structure, rather than extended relational chase) and evaluation methodology (proving that MAPONTO will generate only correct mappings over a well-defined and useful space of relational schemas designed using er2rel).

Important areas for extensions include refinements of our algorithms for complex cases involving is-a hierarchies and missing tables, as well as richer value correspondences. These include dealing with the schema design technique of introducing artificial object identifiers, in cases where there is no natural external key for an entity, or when this is too long.

We are also working on phases (3) of our proposed process: disambiguation between multiple possible semantics. Among others, we expect two additional sources of information to be quite helpful for this task (as well as phase (2)): first, a richer modeling language, supporting at least disjointness/coverage in **is-a** hierarchies, but also more complex axioms as in

OWL ontologies; second, the use of the *data* stored in the relational tables whose semantics we are investigating. For example, queries may be used to check whether complex integrity constraints implied by the semantics of a concept/relationship fail to hold, thereby eliminating some candidate semantics.

Extending this work to XML, and further empirical evaluation on XML data are obvious additional topics that remain on our agenda.

Acknowledgments: We are most grateful to Renée Miller and Yannis Velegrakis for their clarifications concerning Clio, comments on our results, and encouragement. Remaining errors are, of course, our own.

References

[1] J. Barrasa, O. Corcho, and A. Gómez-Pérez. R_2O, An Extensible and Semantically Based Database-to-Ontology Mapping Language. In *SWDB'04*, 2004.

[2] D. Calvanese, G. D. Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Data Integration in Data Warehousing. *J. of Coop. Info. Sys.*, 10(3):237–271, 2001.

[3] M. Dahchour and A. Pirotte. The Semantics of Reifying n-ary Relationships as Classes. In *Proc. ICEIS'02*, pages 580–586, 2002.

[4] 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.

[5] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider. Sweetening Ontologies with DOLCE. In *EKAW'02*, pages 166–181, 2002.

[6] J.-L. Hainaut. *Database Reverse Engineering.* http:// citeseer.ist.psu.edu/ article/ hainaut98database.html, 1998.

[7] S. Handschuh, S. Staab, and R. Volz. On Deep Annotation. In *Proc. WWW'03*, 2003.

[8] J. Heflin. *The Academic Department Ontology.* http://www.daml.org, http://www.daml.org/ontologies/, 2000.

[9] 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 *ISWC'04*, pages 229–243, Nov. 2004.

[10] M. Lenzerini. Data Integration: A Theoretical Perspective. In *PODS'02*, pages 233–246, 2002.

[11] A. Y. Levy, D. Srivastava, and T. Kirk. Data Model and Query Evaluation in Global Information Systems. *J. of Intelligent Info. Sys.*, 5(2):121–143, Dec 1996.

[12] V. M. Markowitz and J. A. Makowsky. Identifying Extended Entity-Relationship Object Structures in Relational Schemas. *IEEE TSE*, 16(8):777–790, August 1990.

[13] E. Mena, V. Kashyap, A. Sheth, and A. Illarramendi. OBSERVER: An Approach for Query Processing in Global Information Systems Based on Interoperation Across Preexisting Ontologies. In *CoopIS'96*, pages 14–25, 1996.

[14] R. Miller, L. M. Haas, and M. A. Hernandez. Schema Mapping as Query Discovery. In *VLDB'00*, pages 77–88, 2000.

[15] OpenCyc. *http://www.opencyc.org.* , 2003.

[16] P. F. Patel-Schneider, P. Hayes, and I. Horrocks. *OWL Web Ontology Language Semantics and Abstract Syntax.* W3C Recommendation, http://www.w3c.org/TR/owl-semantics, 2004.

[17] L. Popa, Y. Velegrakis, R. J. Miller, M. Hernandes, and R. Fagin. Translating Web Data. In *VLDB'02*, pages 598–609, 2002.

[18] M. R. Quillian. Semantic Memory. In *Semantic Information Processing*, pages 227–270. The MIT Press, 1968.

[19] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10:334–350, 2001.

[20] R. Ramakrishnan and M. Gehrke. *Database Management Systems (3rd ed.).* McGraw Hill, 2002.

[21] L. Seligman, A. Rosenthal, P. Lehner, and A. Smith. Data Integration: Where Does the Time Go? *IEEE Data Eng. Bull.*, (3):3–10, 2002.

[22] 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 *IJCAI'01 Wkshp. on Ontologies and Information Sharing*, 2001.