Discovering the Semantics of Relational Tables Through Mappings [$\star$]

Yuan An[1], Alex Borgida[2], and John Mylopoulos[1]

[1] Department of Computer Science, University of Toronto, Canada
{yuana, jm}@cs.toronto.edu
[2] Department of Computer Science, Rutgers University, USA
borgida@cs.rutgers.edu

Abstract. Many problems in Information and Data Management require a semantic account of a database schema. At its best, such an account consists of formulas expressing the relationship ("mapping") between the schema and a formal conceptual model or ontology (CM) of the domain. In this paper we describe the underlying principles, algorithms, and a prototype tool that finds such semantic mappings from relational tables to ontologies, when given as input *simple correspondences* from columns of the tables to datatype properties of classes in an ontology. Although the algorithm presented is necessarily heuristic, we offer formal results showing that the answers returned by the tool are "correct" for relational schemas designed according to standard Entity-Relationship techniques. To evaluate its usefulness and effectiveness, we have applied the tool to a number of public domain schemas and ontologies. Our experience shows that significant effort is saved when using it to build semantic mappings from relational tables to ontologies.

Keywords: Semantics, ontologies, mappings, semantic interoperability.

1 Introduction and Motivation

A number of important database problems have been shown to have improved solutions by using a conceptual model or an ontology (CM) to provide *precise semantics* for a database schema. These[1] include federated databases, data warehousing [2], and information integration through mediated schemas [13,8]. 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 database-resident data (e.g., [10]). In almost all of these cases, semantics of the data is captured by 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 what is required is a *complex* formula, often expressed in logic or a query language [14].

For example, in both the Information Manifold data integration system presented in [13] and the DWQ data warehousing system [2], formulas of the form $T(\overline{X})$:- $\Phi(\overline{X}, \overline{Y})$

[$\star$] This is an expanded and refined version of a research paper presented at ODBASE'05 [1].

[1] For a survey, see [23].

are used to connect a relational data source to a CM expressed in terms of a Description Logic, where $T(\overline{X})$ is a single predicate representing a table in the relational data source, and $\Phi(\overline{X}, \overline{Y})$ is a conjunctive formula over the predicates representing the concepts and relationships in the CM. In the literature, such a formalism is called local-as-view (LAV), in contrast to global-as-view (GAV), where atomic ontology concepts and properties are specified by queries over the database [14].

In all previous work it has been assumed that *humans* specify the mapping formulas – a difficult, time-consuming and error-prone task, especially since the specifier must be familiar with both the semantics of the database schema and the contents of the ontology. As the size and complexity of ontologies increase, it becomes desirable to have some kind of computer tool to assist people in the task. Note that the problem of semantic mapping discovery is superficially similar to that of database schema mapping, however the goal of the later is finding queries/rules for integrating/translating/exchanging the underlying data. Mapping schemas to ontologies, on the other hand, is aimed at understanding the semantics of a schema expressed in terms of a given semantic model. This requires paying special attentions to various semantic constructs in both schema and ontology languages.

We have proposed in [1] a tool that assists users in discovering mapping formulas between relational database schemas and ontologies, and presented the algorithms and the formal results. In this paper, we provide, in addition to what appears in [1], more detailed examples for explaining the algorithms, and we also present proofs to the formal results. Moreover, we show how to handle GAV formulas that are often useful for many practical data integration systems. The heuristics that underlie the discovery process are based on a careful study of standard design process relating the constructs of the relational model with those of conceptual modeling languages. In order to improve the effectiveness of our tool, we assume some user input in addition to the database schema and the ontology. Specifically, inspired by the Clio project [17], we expect the tool user to provide *simple correspondences* between atomic elements used in the database schema (e.g., column names of tables) and those in the ontology (e.g., attribute/"data type property" names of concepts). Given the set of correspondences, the tool is expected to reason about the database schema and the ontology, and to generate a list of candidate formulas for each table in the relational database. Ideally, one of the formulas is the correct one — capturing user intention underlying given correspondences. The claim is that, compared to composing logical formulas representing semantic mappings, it is much easier for users to (i) draw simple correspondences/arrows from column names of tables in the database to datatype properties of classes in the ontology[2] and then (ii) evaluate proposed formulas returned by the tool. The following example illustrates the input/output behavior of the tool proposed.

Example 1.1. An ontology contains concepts (classes), attributes of concepts (datatype properties of classes), relationships between concepts (associations), and cardinality constraints on occurrences of the participating concepts in a relationship. Graphically, we use the UML notations to represent the above information. Figure 1 is an enterprise ontology containing some basic concepts and relationships. (Recall that cardinality

[2] In fact, there exist already tools used in schema matching which help perform such tasks using linguistic, structural, and statistical information (e.g., [4,21]).

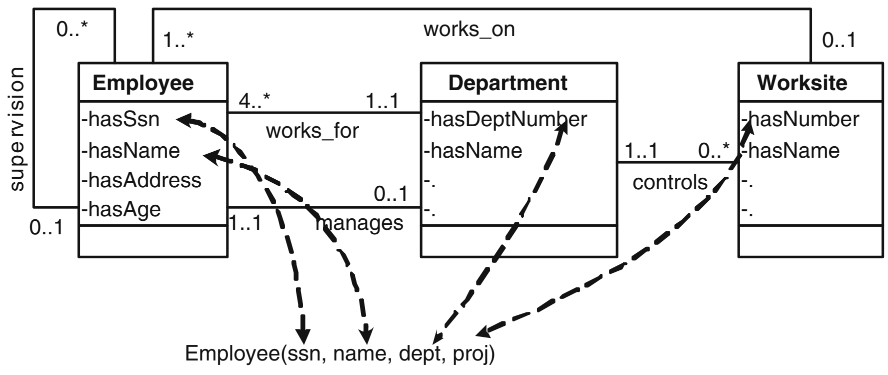


Fig. 1. Relational table, Ontology, and Correspondences

constraints in UML are written at the opposite end of the association: a Department has at least 4 Employees working for it, and an Employee works in one Department.) Suppose we wish to discover the semantics of a relational table $Employee(\underline{ssn}, name, dept, proj)$ with key ssn in terms of the enterprise ontology. Suppose that by looking at column names of the table and the ontology graph, the user draws the simple correspondences shown as dashed arrows in Figure 1. This indicates, for example, that the ssn column corresponds to the $hasSsn$ property of the $Employee$ concept. Using prefixes $\mathcal{T}$ and $\mathcal{O}$ to distinguish tables in the relational schema and concepts in the ontology (both of which will eventually be thought of as predicates), we represent the correspondences as follows:

$\mathcal{T} : Employee.ssn \leftrightsquigarrow \mathcal{O} : Employee.hasSsn$
$\mathcal{T} : Employee.name \leftrightsquigarrow \mathcal{O} : Employee.hasName$
$\mathcal{T} : Employee.dept \leftrightsquigarrow \mathcal{O} : Department.hasDeptNumber$
$\mathcal{T} : Employee.proj \leftrightsquigarrow \mathcal{O} : Worksite.hasNumber$

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

$\mathcal{T}$:Employee$(ssn, name, dept, proj)$:-
  $\mathcal{O}$: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$(x_3,proj)$.

Note that, as explained in [14], the above, admittedly confusing notation in the literature, should really be interpreted as the First Order Logic formula

$(\forall ssn, name, dept, proj)$ $\mathcal{T}$:Employee$(ssn, name, dept, proj) \Rightarrow$
$(\exists x_1, x_2, x_3)$ $\mathcal{O}$:Employee$(x_1) \wedge$...

because the ontology *explains* what is in the table (i.e., every tuple corresponds to an employee), rather than guaranteeing that the table satisfies the closed world assumption (i.e., for every employee there is a tuple in the table). ■

An intuitive (but somewhat naive) solution, inspired by early work of Quillian [20], is based on finding the *shortest* connections between concepts. Technically, this involves

(i) finding the minimum spanning tree(s) (actually Steiner trees[3]) connecting the "corresponded concepts" — those that have datatype properties corresponding to table columns, and then (ii) encoding the tree(s) into formulas. However, in some cases the spanning/Steiner tree may not provide the desired semantics for a table because of known relational schema design rules. For example, consider the relational table $Project$ ($\underline{name}$, $supervisor$), where the column $name$ is the key and corresponds to the attribute $\mathcal{O}$:$Worksite.hasName$, and column $supervisor$ corresponds to the attribute $\mathcal{O}$:$Employee.hasSsn$ in Figure 1. The minimum spanning tree consisting of $Worksite$, $Employee$, and the edge $works\_on$ probably does not match the semantics of table $Project$ because there are multiple $Employee$s working on a $Worksite$ according to the ontology cardinality, yet the table allows only one to be recorded, since $supervisor$ is functionally dependent on $name$, the key. Therefore we must seek a functional connection from $Worksite$ to $Employee$, and the connection will be the manager of the department controlling the worksite. In this paper, we use ideas of standard relational schema design from ER diagrams in order to craft heuristics that systematically uncover the connections between the constructs of relational schemas and those of ontologies. We propose a tool to generate "reasonable" trees connecting the set of corresponded concepts in an ontology. In contrast to the graph theoretic results which show that there may be too many minimum spanning/Steiner trees among the ontology nodes (for example, there are already 5 minimum spanning trees connecting $Employee$, $Department$, and $Worksite$ in the very simple graph in Figure 1), we expect the tool to generate only a small number of "reasonable" trees. These expectations are born out by our experimental results, in Section 6.

As mentioned earlier, our approach is directly inspired by the Clio project [17,18], which developed a successful tool that infers mappings from one set of relational tables and/or XML schemas to another, given just a set of correspondences between their respective attributes. Without going into further details at this point, we summarize the contributions of this work:

- We identify a new version of the data mapping problem: that of *inferring* complex formulas expressing the semantic mapping between relational database schemas and ontologies from simple correspondences.
- We propose an algorithm to find "reasonable" tree connection(s) in the ontology graph. 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 theoretical confidence, we give formal results for a limited class of schemas. We show that if the schema was designed from a CM using techniques well-known in the Entity Relationship literature (which provide a natural semantic mapping and correspondences for each table), then the tool will recover essentially all and only the appropriate semantics. This shows that our heuristics are not just shots in the dark: in the case when the ontology has no extraneous material, and when a table's scheme has not been denormalized, the algorithm will produce good results.

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

– To test the effectiveness and usefulness of the algorithm in practice, we implemented the algorithm in a prototype tool and applied it to a variety of database schemas and ontologies drawn from a number of domains. We ensured that the schemas and the ontologies were developed independently; and the schemas might or might not be derived from a CM using the standard techniques. Our experience has shown that the user effort in specifying complex mappings by using the tool is significantly less than that by manually writing formulas from scratch.

The rest of the paper is structured as follows. We contrast our approach with related work in Section 2, and in Section 3 we present the technical background and notation. Section 4 describes an intuitive progression of ideas underlying our approach, while Section 5 provides the mapping inference algorithm. In Section 6 we report on the prototype implementation of these ideas and experiments with the prototype. Section 7 shows how to filter out unsatisfied mapping formulas by ontology reasoning. Section 8 discusses the issues of generating GAV mapping formulas. Finally, Section 9 concludes and discusses future work.

2 Related Work

The Clio tool [17,18] discovers formal queries describing how target schemas can be populated with data from source schemas. To compare with it, we could view the present work as extending Clio to the case when the source schema is a relational database while the target is an ontology. For example, in Example 1.1, if one viewed the ontology as a relational schema made of unary tables (such as $Employee(x_1)$), binary tables (such as $hasSsn(x_1, ssn)$) and the obvious foreign key constraints from binary to unary tables, then one could in fact try to apply directly the Clio algorithm to the problem. The desired mapping formula from Example 1.1 would not be produced for several reasons: (i) Clio [18] works by taking each table and using a chase-like algorithm to repeatedly extend it with columns that appear as foreign keys referencing other tables. Such "logical relations" in the source and target are then connected by queries. In this particular case, this would lead to logical relations such as $works\_for \bowtie Employee \bowtie Department$, but none that join, through some intermediary, $hasSsn(x_1, ssn)$ and $hasDeptNumber(x_2, dept)$, which is part of the desired formula in this case. (ii) The fact that ssn is a key in the table $\mathcal{T}$:$Employee$, leads us to prefer (see Section 4) a many-to-one relationship, such as $works\_for$, over some many-to-many relationship which could have been part of the ontology (e.g., $\mathcal{O}$:$previouslyWorkedFor$); Clio does not differentiate the two. So the work to be presented here analyzes the key structure of the tables and the semantics of relationships (cardinality, IsA) to eliminate/downgrade *unreasonable* options that arise in mappings to ontologies.

Other potentially relevant work includes *data reverse engineering*, which aims to extract a CM, such as an ER diagram, from a database schema. Sophisticated algorithms and approaches to this have appeared in the literature over the years (e.g., [15,9]). The major difference between data reverse engineering and our work is that we are given an existing ontology, and want to interpret a legacy relational schema in terms of it, whereas data reverse engineering aims to construct a new ontology.

Schema matching (e.g., [4,21]) identifies 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 our mapping inference algorithms.

3 Formal Preliminaries

We do not restrict ourselves to any particular language for describing ontologies in this paper. Instead, we use a generic conceptual modeling language (CML), which contains *common* aspects of most semantic data models, UML, ontology languages such as OWL, and description logics. In the sequel, we use CM to denote an ontology prescribed by the generic CML. Specifically, the language allows the representation of *classes/concepts* (unary predicates over individuals), *object properties/relationships* (binary predicates relating individuals), and *datatype properties/attributes* (binary predicates relating individuals with values such as integers and strings); attributes are single valued in this 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 cardinality constraints, which here allow 1 as lower bounds (called *total* relationships), and 1 as upper bounds (called *functional* relationships).

We shall represent a given CM using a labeled directed graph, called an *ontology graph*. We construct the ontology graph from a CM as follows: We create a concept node labeled with C for each concept C, and an edge labeled with p from the concept node C_1 to the concept node C_2 for each object property p with domain C_1 and range C_2; for each such p, there is also an edge in the opposite direction for its inverse, referred to as p^-. For each attribute f of concept C, we create a separate attribute node denoted as $N_{f,C}$, whose label is f, and add an edge labeled f from node C to $N_{f,C}$.[4] For each **is-a** edge from a subconcept C_1 to a superconcept C_2, we create an edge labeled with *is-a* from concept node C_1 to concept node C_2. For the sake of succinctness, we sometimes use UML notations, as in Figure 1, to represent the ontology 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 concepts C and D will be written in text as $\boxed{\texttt{C}}$ `---p---` $\boxed{\texttt{D}}$. If the relationship p is functional from C to D, we write $\boxed{\texttt{C}}$ `---p->--` $\boxed{\texttt{D}}$. For expressive CMLs such as OWL, we may also connect C to D by p if we find an existential restriction stating that each instance of C is related to *some* instance or *only* instances of D by p.

For relational databases, we assume the reader is familiar with standard notions as presented in [22], for example. We will use the notation $T(\underline{K}, Y)$ to represent a relational table T with columns KY, and key K. If necessary, we will refer to the individual columns in Y using $Y[1], Y[2], \ldots$, and use XY as concatenation of columns. 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

[4] Unless ambiguity arises, we say "node C", when we mean "concept node labeled C".

"attribute" for aspects of the CM.) A foreign key (abbreviated as *f.k.* henceforth) in T is a set of columns F that *references* the key of table T', and imposes a constraint that the projection of T on F is a subset of the projection of T' on $\mathsf{key}(T')$.

In this paper, a *correspondence* $T.c \leftrightsquigarrow D.f$ relates column c of table T to attribute f of concept D. Since our algorithms deal with ontology graphs, formally a correspondence L will be a mathematical relation $L(T, c, D, f, N_{f,D})$, where the first two arguments determine unique values for the last three. This means that we only treat the case when a table column corresponds to single attribute of a concept, and leave to future work dealing with complex correspondences, which may represent unions, concatenations, etc.

Finally, for LAV-like mapping, we use Horn-clauses in the form $T(X)$:- $\Phi(X, Y)$, as described in Section 1, to represent *semantic mappings*, where T is a table with columns X (which become arguments to its predicate), and Φ is a conjunctive formula over predicates representing the CM, with Y existentially quantified, as usual.

4 Principles of Mapping Inference

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

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

Consider the case when $T(\underline{c}, b)$ is a table with key c, corresponding to an attribute f on concept C, and b is a foreign key corresponding to an attribute e on concept B. Then for each value of c (and hence instance of C), T associates at most one value of b (instance of B). Hence the semantic mapping for T should be some formula that acts as a function from its first to its second argument. The semantic trees for such formulas look like functional edges in the ontology, and hence are more reasonable. For example, given table $Dep(\underline{dept}, ssn, \ldots)$, and correspondences
$\mathcal{T}{:}Dep.dept \leftrightsquigarrow \mathcal{O}{:}Department.hasDeptNumber$
$\mathcal{T}{:}Dep.ssn \leftrightsquigarrow \mathcal{O}{:}Employee.hasSsn$
from the table columns to attributes of the ontology in Figure 1, the proper semantic tree uses `manages`$^-$ (i.e., `hasManager`) rather than `works_for`$^-$ (i.e., `hasWorkers`).

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

To deal with such problems, our algorithm works in two stages: first connects the concepts corresponding to key columns into a *skeleton tree*, then connects the rest of the corresponded nodes to the skeleton by functional edges (whenever possible).

We must however also deal with the assumption that the relational schema and the CM were developed independently, which implies that not all parts of the CM are reflected in the database schema. This complicates things, since in building the semantic tree we may need to go through additional nodes, which end up not corresponding to columns of the relational table. For example, consider again the table $Project(\underline{name}, supervisor)$ and its correspondences mentioned in Section 1. Because of the key structure of this table, based on the above arguments we will prefer the functional *path*[5] `controls`$^{-}$.`manages`$^{-}$ (i.e., `controlledBy` followed by `hasManager`), passing through node $Department$, over the shorter path consisting of edge `works_on`, which is not functional. Similar situations arise when the CM contains detailed *aggregation* hierarchies (e.g., $city$ part-of $township$ part-of $county$ part-of $state$), which are abstracted in the database (e.g., a table with columns for $city$ and $state$ only).

We have chosen to flesh out the above principles in a systematic manner by considering the behavior of our proposed algorithm on relational schemas designed from Entity Relationship diagrams — a technique widely covered in undergraduate database courses [22]. (We refer to this er2rel *schema design*.) One benefit of this approach is that it allows us to prove that our algorithm, though heuristic in general, is in some sense "correct" for a certain class of schemas. Of course, in practice such schemas may be "denormalized" in order to improve efficiency, and, as we mentioned, only parts of the CM may be realized in the database. Our algorithm uses the general principles enunciated above even in such cases, with relatively good results in practice. Also note that the assumption that a given relational schema was designed from some ER conceptual model does not mean that given ontology is this ER model, or is even expressed in the ER notation. In fact, our heuristics have to cope with the fact that it is missing essential information, such as keys for weak entities.

To reduce the complexity of the algorithms, which essentially enumerate all trees, and to reduce the size of the answer set, we modify an ontology graph by collapsing multiple edges between nodes E and F, labeled $p_1, p_2, \ldots$ say, into at most three edges, each labeled by a string of the form $'p_{j_1}; p_{j_2}; \ldots'$: one of the edges has the names of all functions from E to F; the other all functions from F to E; and the remaining labels on the third edge. (Edges with empty labels are dropped.) Note that there is no way that our algorithm can distinguish between semantics of the labels on one kind of edge, so the tool offers all of them. It is up to the user to choose between alternative labels, though the system may offer suggestions, based on additional information such as heuristics concerning the identifiers labeling tables and columns, and their relationship to property names.

5 Semantic Mapping Inference Algorithms

As mentioned, our algorithm is based in part on the relational database schema design methodology from ER models. We introduce the details of the algorithm iteratively, by incrementally adding features of an ER model that appear as part of the CM. We assume

[5] One consisting of a sequence of edges, each of which represents a function from its source to its target.

that the reader is familiar with basics of ER modeling and database design [22], though we summarize the ideas.

5.1 ER$_0$: An Initial Subset of ER Notions

We start with a subset, ER$_0$, of ER that supports entity sets E (called just "entity" here), with attributes (referred to by $\mathsf{attribs}(E)$), and binary relationship sets. In order to facilitate the statement of correspondences and theorems, we assume in this section that attributes in the CM have globally unique names. (Our implemented tool does not make this assumption.) An entity is represented as a concept/class in our CM. A binary relationship set corresponds to two properties in our CM, one for each direction. Such a relationship is called *many-many* if neither it nor its inverse is functional. A *strong entity* S has some attributes that act as identifier. We shall refer to these using $\mathsf{unique}(S)$ when describing the rules of schema design. A *weak entity* W has instead $\mathsf{localUnique}(W)$ attributes, plus a functional total binary relationship p (denoted as $\mathsf{idRel}(W)$) to an identifying owner entity (denoted as $\mathsf{idOwn}(W)$).

Example 5.1. An ER$_0$ diagram is shown in Figure 2, which has a weak entity $Dependent$ and three strong entities: $Employee$, $Department$, and $Project$. The owner entity of $Dependent$ is $Employee$ and the identifying relationship is $dependents\_of$. Using the notation we introduced, this means that
$\mathsf{localUnique}(Dependent)$ =$deName$, $\mathsf{idRel}(Dependent)$= $dependents\_of$, $\mathsf{idOwn}(Dependent)$= $Employee$. For the owner entity $Employee$, $\mathsf{unique}(Employee)$= $hasSsn$. ■

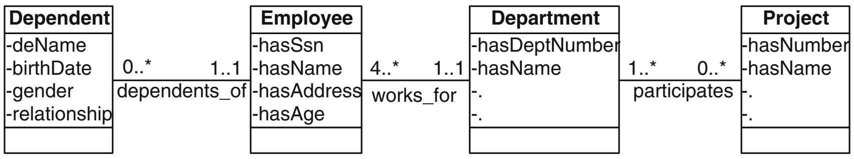


Fig. 2. An ER$_0$ Example

Note that information about multi-attribute keys cannot be represented formally in even highly expressive ontology languages such as OWL. So functions like unique are only used while describing the $\mathsf{er2rel}$ mapping, and are not assumed to be available during semantic inference. The $\mathsf{er2rel}$ design methodology (we follow mostly [15,22]) is defined by two components. To begin with, Table 1 specifies a mapping $\tau(O)$ returning a relational table scheme for every CM component O, where O is either a concept/entity or a binary relationship. (For each relationship exactly one of the directions will be stored in a table.)

In addition to the schema (columns, key, f.k.'s), Table 1 also associates with a relational table $T(V)$ a number of additional notions:

– an *anchor*, which is the central object in the CM from which T is derived, and which is useful in explaining our algorithm (it will be the root of the semantic tree);

Table 1. er2rel Design Mapping

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

– a formula for the semantic mapping for the table, expressed as a formula with head $T(V)$ (this is what our algorithm should be recovering); in the body of the formula, the function hasAttribs(x, Y) returns conjuncts $attr_j(x, Y[j])$ for the individual columns $Y[1], Y[2], \ldots$ in Y, where $attr_j$ is the attribute name corresponded by column $Y[j]$.
– the formula for a predicate identify$_C(x, Y)$, showing how object x in (strong or weak) entity C can be identified by values in Y[6].

Note that τ is defined recursively, and will only terminate if there are no "cycles" in the CM (see [15] for definition of cycles in ER).

Example 5.2. When τ is applied to concept $Employee$ in Figure 2, we get the table $\mathcal{T}$:$Employee$($\underline{hasSsn}$, $hasName$, $hasAddress$, $hasAge$), with the anchor $Employee$, and the semantics expressed by the mapping:
$\mathcal{T}$:Employee($hasSsn$, $hasName$, $hasAddress$, $hasAge$) :-
$\mathcal{O}$:Employee(y), $\mathcal{O}$:hasSsn(y, $hasSsn$), $\mathcal{O}$:hasName(y, $hasName$),
$\mathcal{O}$:hasAddress(y, $hasAddress$), $\mathcal{O}$:hasAge(y, $hasAge$).

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

Its identifier is represented by
identify$_{Employee}(y, hasSsn)$:- $\mathcal{O}$:Employee(y), $\mathcal{O}$:hasSsn($y, hasSsn$).

In turn, $\tau(Dependent)$ produces the table $\mathcal{T}$:$Dependent(\underline{deName, hasSsn}$, $birthDate$,...), whose anchor is $Dependent$. Note that the $hasSsn$ column is a foreign key referencing the $hasSsn$ column in the $\mathcal{T}$:$Employee$ table. Accordingly, its semantics is represented as:
$\mathcal{T}$:Dependent($deName, hasSsn, birthDate, ...$) :-
  $\mathcal{O}$:Dependent(y), $\mathcal{O}$:Employee(w), $\mathcal{O}$:dependents_of(y, w),
  identify$_{Employee}(w, hasSsn)$, $\mathcal{O}$:deName($y, deName$),
  $\mathcal{O}$:birthDate($y, birthDate$) ...
and its identifier is represented as:
identify$_{Dependent}(y, deName, hasSsn)$:-
  $\mathcal{O}$:Dependent(y), $\mathcal{O}$:Employee(w), $\mathcal{O}$:dependents_of(y, w),
  identify$_{Employee}(w, hasSsn)$, $\mathcal{O}$:deName($y, deName$).

τ can be applied similarly to the other objects in Figure 2. $\tau(works\_for)$ produces the table $works\_for(\underline{hasSsn}, hasDeptNumber)$. $\tau(participates)$ generates the table $participates(\underline{hasNumber, hasDeptNumber})$. Please note that the anchor of the table generated by $\tau(works\_for)$ is $Employee$, while no single anchor is assigned to the table generated by $\tau(participates)$. ■

The second step of the er2rel schema design methodology suggests that the schema generated 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=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. (We defer the description of the treatment of null values which can arise in the non-key columns of T_1 appearing in T.) For example, we could merge the table $\tau(Employee)$ with the table $\tau(works\_for)$ in Example 5.2 to form a new table $\mathcal{T}$:$Employee2$ ($\underline{hasSsn}$, $hasName$, $hasAddress$, $hasAge$, $hasDeptNumber$), where the column $hasDeptNumber$ is an f.k. referencing $\tau(Department)$. The semantics of the table is:
$\mathcal{T}$:Employee2($hasSsn, hasName, hasAddress, hasAge, hasDeptNumber$):-
  $\mathcal{O}$:Employee(y), $\mathcal{O}$:hasSsn($y, hasSsn$), $\mathcal{O}$:hasName($y, hasName$),
  $\mathcal{O}$:hasAddress($y, hasAddress$), $\mathcal{O}$:hasAge($y, hasAge$),
  $\mathcal{O}$:Department(w), $\mathcal{O}$:works_for(y, w), $\mathcal{O}$:hasDeptNumber($w, hasDeptNumber$).
Please note that one conceptual model may result in several different relational schemas, since there are choices in which direction a one-to-one relationship is encoded (which entity acts as a key), and how tables are merged. Note also that the resulting schema is in Boyce-Codd Normal Form, if we assume that the only functional dependencies are those that can be deduced from the ER schema (as expressed in FOL).

In this subsection, we assume that the CM has no so-called “recursive” relationships relating an entity to itself, and no attribute of an entity corresponds to multiple columns of any table generated from the CM. (We deal with these in Section 5.3.) Note that by the latter assumption, we rule out for now the case when there are several relationships between a weak entity and its owner entity, such as $hasMet$ connecting $Dependent$ and $Employee$, because in this case $\tau(hasMet)$ will need columns $deName, ssn1$, $ssn2$, with $ssn1$ helping to identify the dependent, and $ssn2$ identifying the (other) employee they met.

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

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

To flesh out the above steps, we begin with the tables created by the standard design process. If a table is derived by the er2rel methodology from an ER_0 diagram, then Table 1 provides substantial knowledge about how to determine the skeleton tree. However, care must be taken when weak entities are involved. The following example describes the right process to discover the skeleton and the anchor of a weak entity table.

Example 5.3. Consider table $\mathcal{T}$:$Dept(\underline{number, univ}, dean)$, with foreign key (f.k.) $univ$ referencing table $\mathcal{T}$:$Univ(\underline{name}, address)$ and correspondences shown in Figure 3. We can tell that $\mathcal{T}$:$Dept$ represents a weak entity since its key has one f.k. as a subset (referring to the strong entity on which $Department$ depends). To find the skeleton and anchor of the table $\mathcal{T}$:$Dept$, we first need to find the skeleton and anchor of the table referenced by the f.k. $univ$. The answer is $University$. Next, we should look for a total functional edge (path) from the correspondent of $number$, which is concept $Department$, to the anchor, $University$. As a result, the link $\boxed{\texttt{Department}}$ `---belongsTo-->-` $\boxed{\texttt{University}}$ is returned as the skeleton, and $Department$ is returned as the anchor. Finally, we can correctly identify the $dean$ relationship as the remainder of the connection, rather than the $president$ relationship, which would have seemed a superficially plausible alternative to begin with.

Furthermore, suppose we need to interpret the table $\mathcal{T}$:$Portal(\underline{dept, univ}, address)$ with the following correspondences:
$\mathcal{T} : Portal.dept \leftrightsquigarrow \mathcal{O} : Department.hasDeptNumber$
$\mathcal{T} : Portal.univ \leftrightsquigarrow \mathcal{O} : University.hasUnivName$
$\mathcal{T} : Portal.address \leftrightsquigarrow \mathcal{O} : Host.hostName$,
where not only is $\{dept, univ\}$ the key but also an f.k. referencing the key of table $\mathcal{T}$:$Dept$. To find the anchor and skeleton of table $\mathcal{T}$:$Portal$, the algorithm first recursively works on the referenced table. This is also needed when the owner entity of a weak entity is itself a weak entity. ■

The following is the function getSkeleton which returns a set of (skeleton, anchor)-pairs, when given a table T and a set of correspondences L from key(T). The function is essentially a recursive algorithm attempting to reverse the function τ in Table 1.

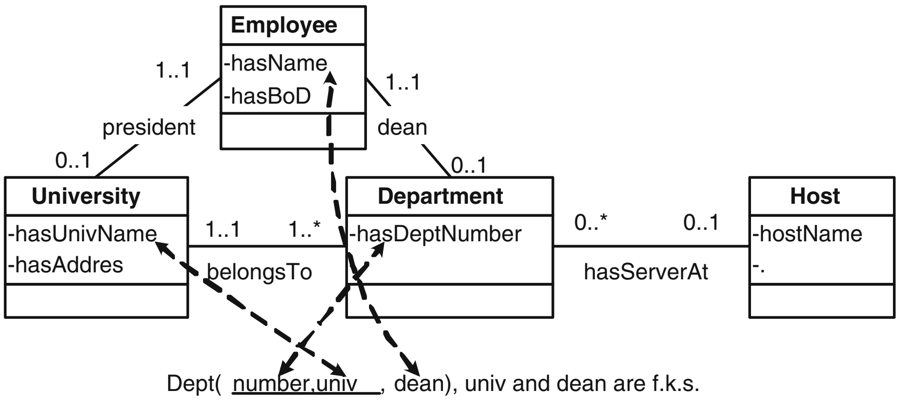


Fig. 3. Finding Correct Skeleton Trees and Anchors

In order to accommodate tables not designed according to er2rel, the algorithm has branches for finding minimum spanning/Steiner trees as skeletons.

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

1. If $n \leq 1$ and onc(key($T$))[7] is just a singleton set $\{C\}$, then return $(C, \{C\})$.[8] /*T is likely
   about a strong entity: base case.*/
2. Else, let $L_i$={$T_i.K_i \leftrightsquigarrow L(T, F_i)$}/*translate corresp's thru f.k. reference.*/;
   compute $(Ss_i, Anc_i)$ = getSkeleton($T_i$, $L_i$), for $i = 1, .., n$.
   (a) If key($T$) $= F_1$, then return ($Ss_1$, $Anc_1$). /*T looks like the table for the functional
       relationship of a weak entity, other than its identifying relationship.*/
   (b) If key($T$)=$F_1A$, where columns $A$ are not part of an f.k. then /*T is possibly a weak
       entity*/
           if $Anc_1 = \{N_1\}$ and onc($A$) $= \{N\}$ such that there is a (shortest) total functional
           path $\pi$ from $N$ to $N_1$, then return (combine[9]($\pi$, $Ss_1$), $\{N\}$). /*N is a weak entity.
           cf. Example 5.3.*/
   (c) Else suppose key($T$) has non-f.k. columns $A[1], \ldots A[m]$, $(m \geq 0)$; let $N_s$={$Anc_i, i =
       1, .., n$} ∪ {onc($A[j]$), $j = 1, .., m$}; find skeleton tree $S'$ connecting the nodes in $N_s$
       where any pair of nodes in $N_s$ is connected by a (shortest) non-functional path; return
       (combine($S'$, $\{Ss_j\}$), $N_s$). /*Deal with many-to-many binary relationships; also the
       default action for non-standard cases, such as when not finding identifying relationship
       from a weak entity to the supposed owner entity. In this case no unique anchor exists.*/
```

[7] onc(X) is the function which gets the set M of concepts corresponded by the columns X.

[8] 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.

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

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

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

```
Function getTree($T$,$L$)
input: table $T$, correspondences $L$ for columns($T$)
output: set of semantic trees [10]
steps:
1. Let $L_k$ be the subset of $L$ containing correspondences from key($T$);
   compute $(S', Anc')$=getSkeleton($T$,$L_k$).
2. If onc(nonkey($T$)) − onc(key($T$)) is empty, then return $(S', Anc')$. /*if all columns correspond to the same set of concepts as the key does, then return the skeleton tree.*/
3. For each f.k. $F_i$ in nonkey($T$) referencing $T_i(K_i)$:
   let $L^i_k = \{T_i.K_i \leftrightsquigarrow L(T, F_i)\}$, and compute $(Ss''_i, Anc''_i)$= getSkeleton($T_i$,$L^i_k$). /*recall that the function $L(T, F_i)$ is derived from a correspondence $L(T, F_i, D, f, N_{f,D})$ such that it gives a concept $D$ and its attribute $f$ ($N_{f,D}$ is the attribute node in the ontology graph.)*/
   find $\pi_i$=shortest functional path from $Anc'$ to $Anc''_i$; let $S$ = combine($S'$, $\pi_i$, $\{Ss''_i\}$).
4. For each column $c$ in nonkey(T) that is not part of an f.k., let $N$ = onc($c$); find $\pi$=shortest functional path from $Anc'$ to $N$; update $S$ := combine($S, \pi$). /*cf. Example 5.4.*/
5. In all cases above asking for functional paths, use a shortest path if a functional one does not exist.
6. Return $S$.
```

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

Example 5.4. In Figure 4, the table $\mathcal{T}$:$Assignment(\underline{emp, proj}, site)$ was originally derived from a CM with the entity $Assignment$ shown on the right-hand side of the vertical dashed line. To interpret it by the CM on the left-hand side, the function getSkeleton, in Step 2.c, returns $\boxed{\texttt{Employee}}$ `---assignedTo---` $\boxed{\texttt{Project}}$ as the skeleton, and no single anchor exists. The set $\{Employee, Project\}$ accompanying the skeleton is

[10] 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.

returned. Subsequently, the function getTree seeks for the shortest functional link from elements in $\{Employee, Project\}$ to $Worksite$ at Step 4. Consequently, it connects $Worksite$ to $Employee$ via $works\_on$ to build the final semantic tree. ■

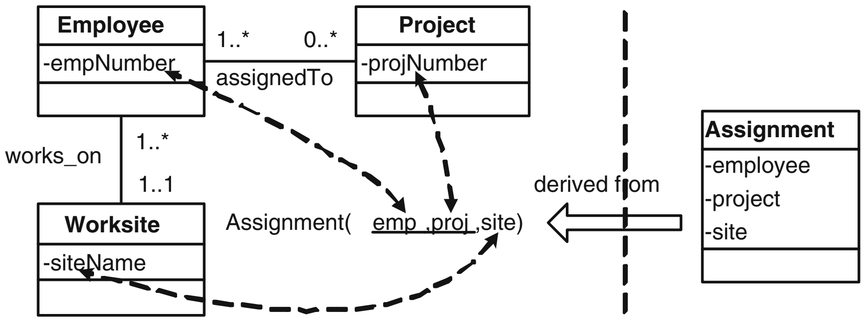


Fig. 4. Independently Developed Table and CM

To get the logic formula from a tree based on correspondence L, we provide 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: Suppose $N$ is the root of $S$. Let $\Psi = true$.
1. if $N$ is an attribute node with label $f$
     find $d$ such that $L(\_, d, \_, f, N) = true$;
     return$(d, true)$. /*for leaves of the tree, which are attribute nodes, return the corresponding column name as the variable and the formula true.*/
2. if $N$ is a concept node with label $C$, then introduce new variable $x$; add conjunct $C(x)$ to $\Psi$;
   for each edge $p_i$ from $N$ to $N_i$ /*recursively get the subformulas.*/
       let $S_i$ be the subtree rooted at $N_i$,
       let $(v_i, \phi_i(Z_i))$=encodeTree$(S_i, L)$,
       add conjuncts $p_i(x, v_i) \wedge \phi_i(Z_i)$ to $\Psi$;
3. return $(x, \Psi)$.
```

Example 5.5. Figure 5 is the fully specified semantic tree returned by the algorithm for the $\mathcal{T}$:$Dept(\underline{number, univ}, dean)$ table in Example 5.3. Taking $Department$ as the root of the tree, function encodeTree generates the following formula:

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

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

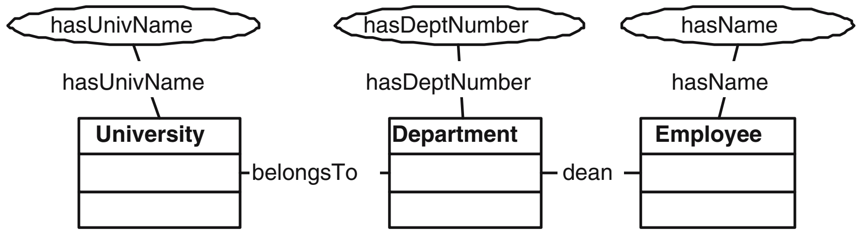


Fig. 5. Semantic Tree For *Dept* Table

Now we turn to the properties of the mapping algorithm. In order to be able to make guarantees, we have to limit ourselves to "standard" relational schemas, since otherwise the algorithm cannot possibly guess the intended meaning of an arbitrary table. For this reason, let us consider only schemas generated by the er2rel methodology from a CM encoding an ER diagram. We are interested in two properties: (1) A sense of "completeness": the algorithm finds the correct semantics (as specified in Table 1). (2) A sense of "soundness": if for such a table there are multiple semantic trees returned by the algorithm, then each of the trees would produce an indistinguishable relational table according to the er2rel mapping. (Note that multiple semantic trees are bound to arise when there are several relationships between 2 entities which cannot be distinguished semantically in a way which is apparent in the table (e.g., 2 or more functional properties from A to B). To formally specify the properties, we have the following definitions.

A *homomorphism* h from the columns of a table T_1 to the columns of a table T_2 is a one-to-one mapping h: $\mathsf{columns}(T_1) \rightarrow \mathsf{columns}(T_2)$, such that (i) $h(c) \in \mathsf{key}(T_2)$ for every $c \in \mathsf{key}(T_1)$; (ii) by convention, for a set of columns F, $h(F[1]F[2]\ldots)$ is $h(F[1])h(F[2])\ldots$; (iii) $h(Y)$ is an f.k. of T_2 for every Y which is an f.k. of T_1; and (iv) if Y is an f.k. of T_1, then there is a homomorphism from the $\mathsf{key}(T_1')$ of T_1' referenced by Y to the $\mathsf{key}(T_2')$ of T_2' referenced by $h(Y)$ in T_2.

Definition 1. *A relational table T_1 is isomorphic to another relational table T_2, if there is a homomorphism from* columns*(T_1) to* columns*(T_2) and vice versa.*

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

To specify the properties of the algorithm, suppose that the correspondence L_{id} is the identity mapping from table columns to attribute names, as set up in Table 1. The following lemma states the interesting property of getSkeleton.

Lemma 1. *Let ontology graph $\mathcal{G}$ encode an ER$_0$ diagram $\mathcal{E}$. Let $T = \tau(C)$ be a relational table derived from an object C in $\mathcal{E}$ according to the* er2rel *rules in Table 1. Given L_{id} from T to $\mathcal{G}$, and L' = the restriction of L_{id} to* key*(T), then* getSkeleton*(T, L') returns (S, Anc) such that,*

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

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

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

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

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

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

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

Theorem 1. *Let ontology graph $\mathcal{G}$ encode an ER$_0$ diagram $\mathcal{E}$. Let table T be part of a relational schema obtained by* er2rel *derivation from $\mathcal{E}$. Given L_{id} from T to $\mathcal{G}$, then some tree S returned by* getTree(T, L_{id}) *has the property that the formula generated by* encodeTree(S, L_{id}) *is logically equivalent to the semantics assigned to T by the* er2rel *design.*

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

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

When $n > 0$, step 1 of getTree constructs a skeleton tree, which represents E by Lemma 1. Step 3 adds edges $f_1, \ldots, f_n$ from E to other entity nodes $E_1, \ldots, E_n$ returned respectively as roots of skeletons for the other foreign keys of T. Lemma 1 also shows that these translate correctly. Steps 4 and 5 cannot apply to tables generated according to er2rel design. So it only remains to note that encodeTree creates the formula for the final tree, by generating conjuncts for $f_1, \ldots, f_n$ and for the non-key attributes of E, and adding these to the formulas generated for the skeleton subtrees at $E_1, \ldots, E_n$.

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

Note that this result is non-trivial, since, as explained earlier, it would not be satisfied by the current Clio algorithm [18], if applied blindly to $\mathcal{E}$ viewed as a relational schema with unary and binary tables. Since getTree may return multiple answers, the following converse "soundness" result is significant.

Theorem 2. *If S' is any tree returned by* getTree(T, L_{id})*, with T, L_{id}, and $\mathcal{E}$ as above in Theorem 1, then the formula returned by* encodeTree(S', L_{id}) *represents the semantics of* some *table T' derivable by* er2rel *design from $\mathcal{E}$, where T' is isomorphic to T.*

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

For the four cases in Table 1, getTree will return a single semantic tree for a table derived from an entity (strong or weak), and possibly multiple semantic trees for a (functional) relationship table. Each of the semantic trees returned for a relationship table is identical to the original ER diagram in terms of the shape and the cardinality constraints. As a result, applying τ to the semantic tree generates a table isomorphic to T.

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

We wish to emphasize that the above algorithms has been designed to deal even with schemas not derived using er2rel from some ER diagram. An application of this was illustrated already in Example 5.4. Another application of this is the use of functional paths instead of just functional edges. The following example illustrates an interesting scenario in which we obtained the right result.

Example 5.6. Consider the following relational table

$T(\underline{personName}, cityName, countryName)$,

where the columns correspond to, respectively, attributes $pname$, $cname$, and $ctrname$ of concepts $Person, City$ and $Country$ in a CM. If the CM contains a path such that $\boxed{\texttt{Person}}$ `-- bornIn ->-` $\boxed{\texttt{City}}$ `-- locatedIn ->-` $\boxed{\texttt{Country}}$, 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,$cityName$),ctrname(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 ER$_1$: Reified Relationships

It is desirable to also have n-ary relationship sets connecting entities, and to allow relationship sets to have attributes in an ER model; we label the language allowing us to model such aspects by ER$_1$. Unfortunately, these features are not directly supported in most CMLs, such as OWL, which only have binary relationships. Such notions must instead be represented by *"reified relationships"* [3] (we use an annotation * to indicate the reified relationships in a diagram): concepts whose instances represent tuples, connected by so-called "roles" to the tuple elements. So, if $Buys$ relates $Person$, $Shop$ and $Product$, through roles $buyer$, $source$ and $object$, then these are explicitly represented as (functional) binary associations, as in Figure 6. And a relationship attribute, such as when the buying occurred, becomes an attribute of the $Buys$ concept, such as $whenBought$.

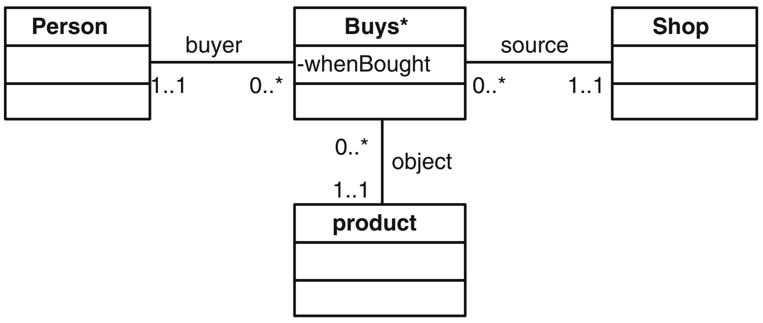


Fig. 6. N-ary Relationship Reified

Unfortunately, reified relationships cannot be distinguished reliably from ordinary entities in normal CMLs based on purely formal, syntactic grounds, yet they need to be treated in special ways during semantic recovery. For this reason we assume that they can be distinguished on *ontological grounds*. For example, in Dolce [7], they are subclasses of top-level concepts $Quality$ and $Perdurant/Event$. For a reified relationship R, we use functions roles(R) and attribs(R) to retrieve the appropriate (binary) properties.

The er2rel design τ of relational tables for reified relationships is an extension of the treatment of binary relationships, and is shown in Table 2. As with entity keys, we are unable to capture in CM situations where some subset of more than one roles uniquely identifies the relationship. The er2rel design τ on ER$_1$ also admits the merge operation on tables generated by τ. Merging applies to an entity table with other tables of some functional relationships involving the same entity. In this case, the merged semantics is

Table 2. er2rel Design for Reified Relationship

ER model object O	**Relational Table $\tau(O)$**
Reified Relationship R	*columns:* $ZX_1 \dots X_n$
if there is a functional	*primary key:* X_1
role r_1 for R	*f.k.'s:* $X_1, \dots, X_n$
$\boxed{E_1}$ `--<- r1 ->--` $\boxed{R}$	*anchor:* R
`--- rj ->--` $\boxed{E_j}$	*semantics:* $T(ZX_1 \dots X_n)$:- $R(y)$,$E_i(w_i)$, hasAttribs(y, Z), $r_i(y, w_i)$,
let Z=attribs(R)	identify$_{E_i}(w_i, X_i), \dots$
X_i=key($\tau(E_i)$)	*identifier:* identify$_R(y, X_1)$:- $R(y)$, $E_1(w)$, $r_1(y, w)$,
where E_i fills role r_i	identify$_{E_1}(w, X_1)$.
Reified Relationship R	*columns:* $ZX_1 \dots X_n$
if $r_1, \dots, r_n$ are roles of R	*primary key:* $X_1 \dots X_n$
let Z=attribs(R)	*f.k.'s:* $X_1, \dots, X_n$
X_i=key($\tau(E_i)$)	*anchor:* R
where E_i fills role r_i	*semantics:* $T(ZX_1 \dots X_n)$:- $R(y)$,$E_i(w_i)$, hasAttribs(y, Z), $r_i(y, w_i)$,
	identify$_{E_i}(w_i, X_i), \dots$
	identifier: identify$_R(y, \dots X_i \dots)$:- $R(y), \dots E_i(w_i)$, $r_i(y, w_i)$,
	identify$_{E_i}(w_i, X_i)$,...

the same as that of merging tables obtained by applying τ to ER$_0$, with the exception that some functional relationships may be reified.

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

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

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

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

In addition, when traversing the ontology graph for finding shortest paths in both functions, we need to recalculate the lengths of paths when reified relationship nodes are present. Specifically, a path of length 2 passing through a reified relationship node should be counted as a path of length 1, because a reified binary relationship could have been eliminated, leaving a single edge.[11] Note that a semantic tree that includes a reified relationship node is valid only if all roles of the reified relationship have been included in the tree. Moreover, if the reified relation had attributes of its own, they would show up as columns in the table that are not part of any foreign key. Therefore, a filter is required at the last stage of the algorithm:

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

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

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

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

5.3 Replication

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

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

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

As far as our previous theorems, one can prove that by making copies of an entity E (say E and E_{copy}), and also replicating its attributes and participating relationships, one obtains an ER diagram from which one can generate isomorphic tables with identical semantics, according to the er2rel mapping. This will hold true as long as the predicate used for **both** E and E_{copy} is $E(\_)$; similarly, we need to use the same predicate for the copies of the attributes and associations in which E and E_{copy} participate.

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

Example 5.7. An educational department in a provincial government records the transfers of students between universities in its databases. A student is a weak entity depending for identification on the university in which the student is currently registered. A transfered student must have registered in another university before transferring. The table $\mathcal{T}$:$Transferred(\underline{sno, univ}, sname)$ records who are the transferred students, and their name. The table $\mathcal{T}$:$previous(\underline{sno, univ}, pUniv)$ stores the information about the $previousUniv$ relationship. A CM is depicted in Figure 7. To discover the seman-

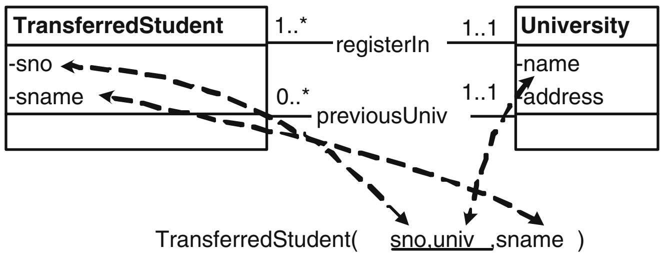


Fig. 7. A Weak Entity and Its Owner Entity

tics of table $\mathcal{T}$:$Transferred$, we link the columns to the attributes in the CM as shown in Figure 7. One of the skeletons returned by the algorithm for the $\mathcal{T}$:$Transferred$ will be $\boxed{\texttt{TransferredStudent}}$ `--- previousUniv ->---` $\boxed{\texttt{University}}$. But the design resulting from this according to the er2rel mapping is not isomorphic to key($Transferred$), since $previousUniv$ is not the identifying relationship of the weak entity $TransferredStudent$. ■

From above example, we can see that the problem is the inability of CMLs such as UML and OWL to fully capture notions like "weak entity" (specifically the notion of identifying relationship), which play a crucial role in ER-based design. We expect such cases to be quite rare though – we certainly have not encountered any in our example databases.

5.4 Extended ER: Adding Class Specialization

The ability to represent subclass hierarchies, such as the one in Figure 8 is a hallmark of CMLs and modern so-called Extended ER (EER) modeling.

Almost all textbooks (e.g., [22]) describe several techniques for designing relational schemas in the presence of class hierarchies

1. Map each concept/entity into a separate table following the standard er2rel rules. This approach requires two adjustments: First, subclasses must inherit identifying attributes from a single super-class, in order to be able to generate keys for their tables. Second, in the table created for an immediate subclass C' of class C, its key $\mathsf{key}(\tau(C'))$ should also be set to reference as a foreign key $\tau(C)$, as a way of maintaining inclusion constraints dictated by the is-a relationship.
2. Expand inheritance, so that *all* attributes and relations involving a class C appear on all its subclasses C'. Then generate tables as usual for the subclasses C', though not for C itself. This approach is used only when the subclasses cover the superclass.
3. Some researchers also suggest a third possibility: "Collapse up" the information about subclasses into the table for the superclass. This can be viewed as the result of $\mathsf{merge}(T_C, T_{C'})$, where $T_C(K, A)$ and $T_{C'}(K, B)$ are the tables generated for C and its subclass C' according to technique (1.) above. In order for this design to be "correct", [15] requires that $T_{C'}$ not be the target of any foreign key references (hence not have any relationships mapped to tables), and that B be non-null (so that instances of C' can be distinguished from those of C).

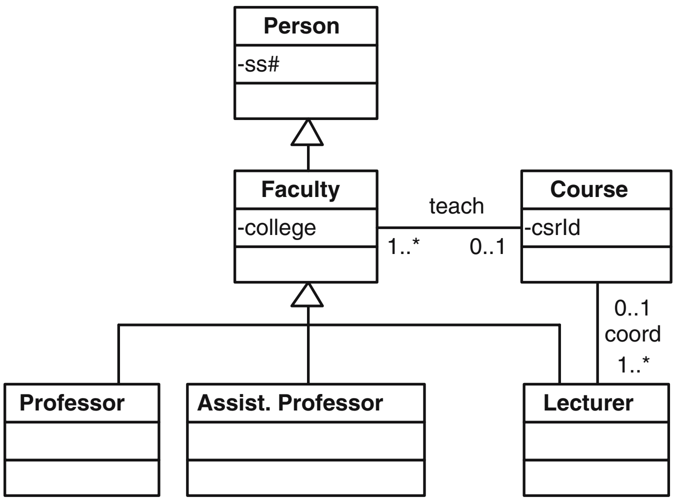


Fig. 8. Specialization Hierarchy

The use of the key for the root class, together with inheritance and the use of foreign keys to also check inclusion constraints, make many tables highly ambiguous. For example, according to the above, table $T(\underline{ss\#}, crsId)$, with $ss\#$ as the key and a foreign key referencing T', could represent at least
(a) $Faculty$ teach $Course$
(b) $Lecturer$ teach $Course$
(c) $Lecturer$ coord $Course$.
This is made combinatorially worse by the presence of multiple and deep hierarchies (e.g., imagine a parallel $Course$ hierarchy), and the fact that not all ontology concepts

are realized in the database schema, according to our scenario. For this reason, we have chosen to deal with some of the ambiguity by relying on users, during the establishment of correspondences. Specifically, the user is supposed to provide a correspondence from column c to attribute f *on the lowest class whose instances provide data appearing in the column*. Therefore, in the above example of table $T(\underline{ss\#}, crsId)$, $ss\#$ should be set to correspond to ssn on $Faculty$ in case (a), while in cases (b) and (c) it should correspond to $ss\#$ on $Lecturer$. This decision was also prompted by the CM manipulation tool that we are using, which automatically expands inheritance, so that $ss\#$ appears on all subclasses.

Under these circumstances, in order to deal appropriately with designs (1.) and (2.) above, we do not need to modify our earlier algorithm in any way, as long as we first expand inheritance in the graph. So the graph would show $\boxed{\texttt{Lecturer}}$ `-- teaches; coord ->-` $\boxed{\texttt{Course}}$ in the above example, and $Lecturer$ would have all the attributes of $Faculty$.

To handle design (3.), we add to the graph an actual edge for the inverse of the **is-a** relation: a functional edge labeled $alsoA$, with lower-bound 0; e.g., $\boxed{\texttt{Faculty}}$ `--- alsoA ->---` $\boxed{\texttt{Lecturer}}$. It is then sufficient to allow in getTree for functional paths between concepts to include `alsoA` edges; e.g., $Faculty$ can now be connected to $Course$ through path `alsoA` followed by `coord`. The `alsoA` edge is translated into the identity predicate, and it is assigned cost zero in evaluating a functional path mixed with `alsoA` edge and other ordinary functional edges.[12]

In terms of the properties of the algorithm we have been considering so far, the above three paragraphs have explained that among the answers returned by the algorithm will be the correct one. On the other hand, if there are multiple results returned by the algorithm, as shown in Example 5.7, some semantic trees may not result in isomorphic tables to the original table, if there are more than one total functional relationships from a weak entity to its owner entity.

5.5 Outer Joins

The observant reader has probably noticed that the definition of the semantic mapping for $T = \text{merge}(T_E, T_p)$, where $T_E(K, V)$:- $\phi(K, V)$ and $T_p(K, W)$:- $\psi(K, W)$, was not quite correct: $T(\underline{K}, V, W)$:-$\phi(K, V)$,ψ (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.

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

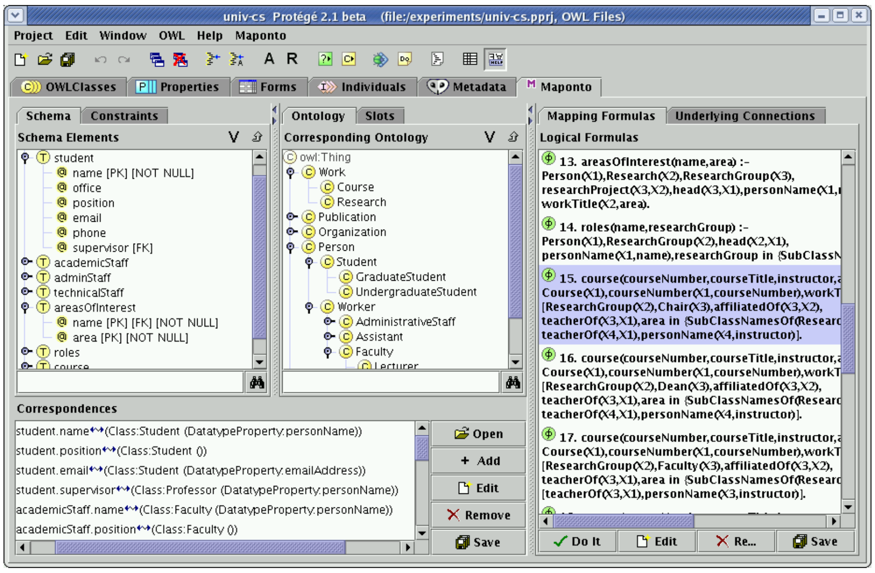
6 Implementation and Experimentation

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

Implementation. We have implemented the MAPONTO tool as a third-party plugin of the well-known KBMS Protégé[13] which is an open platform for ontology modeling and knowledge acquisition. As OWL becomes the official ontology language of the W3C, intended for use with Semantic Web initiatives, we use OWL as the CML in the tool. This is also facilitated by the Protégé's OWL plugin [12], which can be used to edit OWL ontologies, to access reasoners for them, and to acquire instances for semantic markup. The MAPONTO plugin is implemented as a full-size user interface tab that takes advantage of the views of Protégé user interface. As shown in Figure 9, users can choose database schemas and ontologies, create and manipulate correspondences, generate and edit candidate mapping formulas and graphical connections, and produce and save the final mappings into designated files. In addition, there is a library of other Protégé plugins that visualize ontologies graphically and manage ontology versions. Those plugins sustain our goal of providing an interactively intelligent tool to database administrators so that they may establish semantic mappings from the database to ontologies more effectively.

Schemas and Ontologies. Our test data were obtained from various sources, and we have ensured that the databases and ontologies were developed independently. The test data are listed in Table 3. They include the following databases: the Department of Computer Science database in the University of Toronto; the VLDB conference database; the DBLP computer science bibliography database; the COUNTRY database appearing in one of reverse engineering papers [11] (Although the *country* schema is not a real-world database, it appears as a complex experimental example in [11], and has some reified relationship tables, so we chose it to test this aspect of our algorithm); and the test schemas in OBSERVER [16] project. For the ontologies, our test data include: the academic department ontology in the DAML library; the academic conference ontology from the SchemaWeb ontology repository; the bibliography ontology in the library of the Stanford's Ontolingua server; and the CIA factbook ontology. Ontologies are described in OWL. For each ontology, the number of links indicates the number of edges in the multi-graph resulted from object properties. We have made all these schemas and ontologies available on our web page: www.cs.toronto.edu/ ˜yuana/research /maponto/relational/testData.html.

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



```
student.name↔(Class:Student (DatatypeProperty:personName))
student.position↔(Class:Student ())
student.email↔(Class:Student (DatatypeProperty:emailAddress))
student.supervisor↔(Class:Professor (DatatypeProperty:personName))
academicStaff.name↔(Class:Faculty (DatatypeProperty:personName))
academicStaff.position↔(Class:Faculty ())
```

```
13. areasOfInterest(name,area) :-
Person(X1),Research(X2),ResearchGroup(X3),
researchProject(X3,X2),head(X3,X1),personName(X1,
workTitle(X2,area).

14. roles(name,researchGroup) :-
Person(X1),ResearchGroup(X2),head(X2,X1),
personName(X1,name),researchGroup in {SubClassN

15. course(courseNumber,courseTitle,instructor,a
Course(X1),courseNumber(X1,courseNumber),workT
[ResearchGroup(X2),Chair(X3),affiliatedOf(X3,X2),
teacherOf(X3,X1),area in {SubClassNamesOf(Researc
teacherOf(X4,X1),personName(X4,instructor)].

16. course(courseNumber,courseTitle,instructor,a
Course(X1),courseNumber(X1,courseNumber),workT
[ResearchGroup(X2),Dean(X3),affiliatedOf(X3,X2),
teacherOf(X3,X1),area in {SubClassNamesOf(Researc
teacherOf(X4,X1),personName(X4,instructor)].

17. course(courseNumber,courseTitle,instructor,a
Course(X1),courseNumber(X1,courseNumber),workT
[ResearchGroup(X2),Faculty(X3),affiliatedOf(X3,X2),
teacherOf(X3,X1),area in {SubClassNamesOf(Researc
[teacherOf(X3,X1),personName(X3,instructor)].
```

Fig. 9. MAPONTO Plugin of Protege

Table 3. Characteristics of Schemas and Ontologies for the Experiments

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

Results and Experience. To evaluate our tool, we sought to understand whether the tool could produce the intended mapping formula if the simple correspondences were given. We were especially concerned with the number of formulas presented by the tool for users to sift through. Further, we wanted to know whether the tool was still useful if the correct formula was not generated. In this case, we expected that a user could more easily debug a generated formula to reach the correct one instead of creating it from scratch. A summary of the experimental results are listed in Table 4 which shows the average size of each relational table schema in each database, the average number of candidates generated, and the average time for generating the candidates. Notice that the number of candidates is the number of semantic trees obtained by the algorithm. Also, a single edge of an semantic tree may represent the multiple edges between two nodes, collapsed using our $p;q$ abbreviation. If there are m edges in a semantic tree and each edge has n_i $(i = 1, .., m)$ original edges collapsed, then there are $\prod_i^m n_i$ original semantic trees. We show below a formula generated from such a collapsed semantic tree:TaAssignment($courseName, studentName$) :-

Course(x_1), GraduateStudent(x_2), **[hasTAs;takenBy]**(x_1,x_2),
workTitle(x_1,$courseName$), personName(x_2,$studentName$).

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

Table 4. Performance Summary for Generating Mappings from Relational Tables to Ontologies

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

Table 4 indicates that MAPONTO only presents a few mapping formulas for users to examine. This is due in part to our compact representation of parallel edges between two nodes shown above. To measure the overall performance, we manually created the mapping formulas for all the 36 tables and compared them to the formulas generated by the tool. We observed that the tool produced correct formulas for 31 tables. This demonstrates that the tool is able to infer the semantics of many relational tables occurring in practice in terms of an independently developed ontology.

We were also interested in the usefulness of the tool in those cases where the formulas generated were not the intended ones. For each such formula, we compared it to the manually generated correct one, and we used a very coarse measurement to record how much effort it would take to "debug" the generated formula: the number of changes of predicate names in a formula. For example, the tool generated the following formula for the table $Student(name, office, position, email, phone, supervisor)$:

Student(x_1), emailAddress(x_1,$email$), personName(x_1,$name$), Professor(x_2),
Department(x_3), head(x_3,x_2), affiliatedOf(x_3,x_1),
personName(x_2, $supervisor$)... (1)

If the intended semantics for the above table columns is:

Student(x_1), emailAddress(x_1,$email$), personName(x_1,$name$), Professor(x_2),
ResearchGroup(x_3), head(x_3,x_2), affiliatedOf(x_3,x_1),
personName(x_2, $supervisor$)... (2)

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

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

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

(i) *Absence of tables which should be present according to* er2rel. For example, we expect the connection $\boxed{\texttt{Person}}$ `-- researchInterest ---` $\boxed{\texttt{Research}}$ to be returned for the table $AreaOfInterest(\underline{name}, \underline{area})$. However, MAPONTO returned $\boxed{\texttt{Person}}$ `-<- headOf ---` $\boxed{\texttt{ResearchGroup}}$ `-<- researchProject ---` $\boxed{\texttt{Research}}$, because there was no table for the concept $Research$ in the schema, and so MAPONTO treated it as a weak entity table. Such problems are caused, among others, by the elimination of tables that represent finite enumerations, or ones that can be recovered by projection from tables representing total many-to-many relationships. These pose an important open problem for now.

(ii) *Mapping formula requiring selection.* The table $European(\underline{country}, gnp)$ 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 [17].

(iii) *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 not surprisingly, could not add the requirement that $otherAuthors$ is really the concatenation of all but the first author.

7 Filtering Mappings Through Ontology Reasoning

Rich ontologies provide a new opportunity for eliminating "unreasonable" mappings. For example, suppose the ontology specifies that in a library, once a book is reserved for an event, it cannot be borrowed by a person. In this case, a candidate semantic formula such as

$$\text{Book}(x), \text{borrow}(x, y), \text{Person}(y), \text{reservedFor}(x, z), \text{Event}(z)$$

can be eliminated, since no objects x can satisfy it[14].

When ontologies, which include constraints such as the one about $borrowing$ and $reservedFor$, are expressed in OWL, one can use OWL reasoning to detect these problems. To do so, one first translates the semantic tree into an OWL concept, and then checks it for (un)satisfiability in the context of the ontology axioms, using the standard reasoning algorithms for Description Logics.

[14] Maybe a relationship like $contactAuthor(x, y)$, different from $borrow(x, y)$, needs to be used.

For example, the above formula is equivalent to the OWL concept:

```
<owl:intersectionOf>
    <owl:Class rdf:about="#Book"/>
    <owl:Restriction>
       <owl:onProperty rdf:resource=#borrow/>
       <owl:someValuesFrom rdf:resource="#Person"/>
    </owl:Restriction>
    <owl:Restriction>
       <owl:onProperty rdf:resource=#reservedFor/>
       <owl:someValuesFrom rdf:resource="#Event"/>
    </owl:Restriction>
</owl:intersectionOf>
```

The algorithm for performing this translation in general, encodeTreeAsConcept(S), is almost identical to encodeTree, except that the recursive calls return OWL concepts C_i, which lead to conjuncts of the form restriction(p_i, someValuesFrom(C_i)):

```
Function encodeTreeAsConcept($S$)
input: subtree $S$ of ontology graph
output: abstract syntax of OWL concept logically equivalent to the FOL formula
encodeTree($S, L$)
steps: Suppose $N$ is the root of $S$.
1. if $N$ is an attribute node with label $f$
       return restriction($f$,minCardinality(1)). /*for leaves of the tree, which are attribute
nodes, just ensure that the attribute is present.*/
2. if $N$ is a concept node with label $C$, then initialize $\Psi$ to be intersectionOf($C$);
   for each edge $p_i$ from $N$ to $N_i$ /*recursively get the restrictions */
     let $S_i$ be the subtree rooted at $N_i$;
     let $\phi_i$=encodeTreeAsConcept($S_i$);
     add to $\Psi$ a someValuesFrom($\phi_i$) restriction on $p_i$.
3. return $\Psi$.
```

The ontologies we have found so far are unfortunately not sufficiently rich to demonstrate the usefulness of this idea.

8 Finding GAV Mappings

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

To illustrate the idea, consider Example 1.1, from Section 1, where the semantic mapping we proposed was

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

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

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

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

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

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

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

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

$\mathcal{O}$:works_on(ff(ssn),gg($dept$)) :- $\mathcal{T}$:Employee(ssn,_,$dept$,_).

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

$(\forall ssn, name, dept, proj)$ $\mathcal{T}$:Employee($ssn, name, dept, proj$) $\Rightarrow$
$(\exists x, y, z)$ $\mathcal{O}$:Employee(x)$\wedge$ $\mathcal{O}$:hasSsn(x,ssn) $\wedge$ $\mathcal{O}$:hasName(x,$name$) $\wedge$
$\mathcal{O}$:Department(y) $\wedge$ $\mathcal{O}$:hasDeptNumber(y,$dept$) $\wedge$ $\mathcal{O}$:works_for(x,y) $\wedge$
$\mathcal{O}$:Worksite(z) $\wedge$ $\mathcal{O}$:works_on(x,z) $\wedge$ $\mathcal{O}$:hasNumber(z,$proj$).

The above formula can be Skolemized to eliminate the existential quantifiers to yield[15]:

$(\forall ssn, name, dept)$ $\mathcal{T}$:Employee($ssn, name, dept$) $\Rightarrow$
$\mathcal{O}$:Employee(f($ssn, name, dept$)) $\wedge$ $\mathcal{O}$:hasSsn(f($ssn, name, dept$),ssn) $\wedge$
$\mathcal{O}$:hasName(f($ssn, name, dept$),$name$) $\wedge$ $\mathcal{O}$:Department(g($ssn, name, dept$)) $\wedge$
$\mathcal{O}$:hasDeptNumber(g($ssn, name, dept$),$dept$)$\wedge$
$\mathcal{O}$:works_for(f($ssn, name, dept$),g($ssn, name, dept$)).

This implies logically a collection of formulas, including

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

Note however that different tables, such as $\mathcal{T}$:$manages(\underline{ssn, dept})$ say, introduce different Skolem functions, as in :

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

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

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

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

This implies, among others, that
$(\forall ssn, name, dept)\, \mathsf{f}(ssn, name, dept) = \mathsf{h}(ssn, dept)$.
So we need to answer queries over the ontology using all such axioms.

A final, important connection to make in this case is with the research on answering queries using views [6]: The semantic mappings found by the earlier algorithms in this paper can be regarded as view definitions for each relational tables, using conjunctive queries over ontology predicates ("tables"). What we are seeking in this section is answers to queries phrased in terms of the ontology predicates, but rephrased in terms of relational tables, where the data instances reside — which is exactly the problem of query answering using views. The kind of rules we proposed earlier in this section are known as "inverse rules" [19], and in fact Duschka and Levy [5] even deal (implicitly) with the alias problem we mentioned above by their solution to the query answering problem in the presence of functional dependencies: keys functionally determine the rest of the columns in the table.

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

9 Conclusion and Future Work

We have proposed a heuristic algorithm for inferring semantic mapping formulas between relational tables and ontologies starting from simple correspondences. Our algorithm relies on information from the database schema (key and foreign key structure) and the ontology (cardinality restrictions, **is-a** hierarchies). Theoretically, our algorithm infers all and only the relevant semantics if a relational schema was generated using standard database design principles. In practice, our experience working with independently developed schemas and ontologies has shown that significant effort can be saved in specifying the LAV mapping formulas.

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

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. We also deeply appreciate the reviewers' careful readings and constructive comments.

References

1. Y. An, A. Borgida, and J. Mylopoulos Inferring Complex Semantic Mappings between Relational Tables and Ontologies from Simple Correspondences. In *ODBASE'05*, pages 1152-1169, 2005.

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 *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. O. M. Duschka and A. Y. Levy. Recursive Plans for Information Gathering. In *IJCAI'97*, pages 778-784, 1997.
6. A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270-294, 2001.
7. A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider. Sweetening Ontologies with DOLCE. In *EKAW'02*, pages 166–181, 2002.
8. F. Goasdoue et al. Answering queries using views: A KRDB perspective for the semantic web. *ACM TOIT*, 4(3), 2004.
9. J.-L. Hainaut. *Database Reverse Engineering*. http:// citeseer.ist.psu.edu/ article/ hainaut98database.html, 1998.
10. S. Handschuh, S. Staab, and R. Volz. On Deep Annotation. In *Proc. WWW'03*, 2003.
11. P. Johannesson. A method for transforming relational schemas into conceptual schemas. In *ICDE*, pages 190–201, 1994.
12. 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 *ISWC2004*, Nov. 2004.
13. 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.
14. A. Y. Levy. Logic-Based Techniques in Data Integration. In Jack Minker (ed), *Logic Based Artificial Intelligence.*, Kluwer Publishers, 2000
15. V. M. Markowitz and J. A. Makowsky. Identifying Extended Entity-Relationship Object Structures in Relational Schemas. *IEEE TSE*, 16(8):777–790, August 1990.
16. 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.
17. R. Miller, L. M. Haas, and M. A. Hernandez. Schema Mapping as Query Discovery. In *VLDB'00*, pages 77–88, 2000.
18. L. Popa, Y. Velegrakis, R. J. Miller, M. Hernandes, and R. Fagin. Translating Web Data. In *VLDB'02*, pages 598–609, 2002.
19. Xiaolei Qian. Query Folding. In *Proc. ICDE*, 48-55, 1996.
20. M. R. Quillian. Semantic Memory. In *Semantic Information Processing*. Marvin Minsky (editor). 227-270. The MIT Press. 1968.
21. E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10:334–350, 2001.
22. R. Ramakrishnan and M. Gehrke. *Database Management Systems (3rd ed.)*. McGraw Hill, 2002.
23. 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 Workshop. on Ontologies and Information Sharing*, 2001.