Semantic Query Transformation using Ontologies

Chokri Ben Necib
Johann-Christoph Freytag
Department of Computer Science
Humboldt-Universität zu Berlin
Berlin; Germany
{necib, freytag}@dbis.informatik.hu-berlin.de

Abstract

Traditional approaches to query processing aim at rewriting a given query into another more efficient one that uses less time and/or resources during the execution. There by, the rewritten query must be equivalent *to the initial one, i.e., it must provide the same result. However, rewriting queries in equivalent ways do not always satisfy the user's needs, in particular when the user does not receive any answer at all. In this paper we propose a new approach for query processing which allows to rewrite a query into another one which is not necessary equivalent but can provide more meaningful result satisfying the user's intention. For this purpose, we illustrate how semantic knowledge in form of* ontologies *could be effectively used. We develop a set of rewriting rules which rely on semantic information extracted form the ontology associated with the database. In addition, we discuss features of the necessary mappings between the ontology and its underlying database.*

1. Introduction

The problem of *query rewriting* has been addressed by several approaches to query processing including query optimization and answering queries using views. The basic idea of query optimization is to rewrite a given query into another equivalent one that uses less time and/or resources during the execution. Two queries are considered to be equivalent if they provide the same result. In centralized database systems, in particular, query rewriting takes place while generating an execution plan for the query. There by, the query optimizer attempts to find a plan, among other equivalent plans, which can be evaluated against the database with minimal costs. For this purpose, one of the approaches that can be applied is the use of *semantic query optimizations* (SQO) [5, 11, 3, 1]. SQO-techniques are based on the use of semantic rules for the rewriting process. Such rules are derived from various forms of semantic knowledge about the database content such as integrity constraints or additional sources.

On the other hand, answering queries using views deals with the problem of how to select useful views, which are defined over the database schema, in order to transform an input query into another one using only such views [13]. For instance, Levy and his group proposed an approach to transforming a given query by replacing its expression with a new expression containing materialized view definitions such that the transformed query is equivalent to the inial one and could be executed more efficiently [14]. In fact, using materialized views can speed up query processing since a part of the query computation have been already done while computing the views.

The previous approaches state the condition that the transformed queries must be equivalent to the original ones. We argue that this condition should be relaxed in many situations where a user does not need only ”exact” answers for their queries but those answers that meet his intention. We define an exact answer as being the set of database items which literally match the terms of query predicates. In fact, responses to queries may not provide information the user really wants. A user may need additional information or even different information than the query requests [8]. Furthermore, he may prefer an alternative answer to his queries over not receiving any answer at all [12]. There can be several arguments. First, the information stored in databases are usually captured in natural languages. This leads to several variations in expression of the same concept (synonym problem). For instance, assuming that a user requests a database containing items of products to inquire for information about the product ”computer”. Thus, the user would not obtain all instances from the database unless he know in advance that the database contains synonyms of ”computer” such as ”data processor” or ”computing machine” [4]. Moreover, languages introduces multiple mean-

ings of the same expression (homonym problem). For instance, the word "bank" has different meanings. It means a kind of seats for users who search for furniture whereas it means a coins container for users who want to keep money in small devices at home [4]. These problems might affect the query results when formulating queries using certain terms. Second, there might be different ways to formulate a query for characterizing the same entities in the database. In fact, the user's perception of real world things might not match exactly that of the database designer who registers information about these things. In our approach we address this issue as a semantic problem rather than a pattern matching problem.

In this paper we propose an alternative approach for query processing which allows a relaxation of the equivalence assumption. Our goal is to process a query at both syntactical and semantical level. A query can be rewritten into a new one which is not necessary equivalent but can provide more meaningful answer satisfying the user's intention. This answer may then contain more or less database instances than those returned by the old query but it could fully meet the intention of the user. To this end, we propose to associate the database with a semantic knowledge source in the form of an *ontology* in order to capture additional semantics of its content. Based on these semantics we develop a set of transformation rules for the rewriting process. We believe that existing DBMSs need additional supports to overcome semantic problems during query processing. The use of ontologies is a promising candidate for this task.

Shortly, we consider the following issue:
Given a database DB, an ontology O and a user query Q, find a transformed query Q' of Q by using O such that Q' returns more meaningful answer to the user than Q.

The remainder of this paper is organized as follows. First, we give a definition of an ontology and its representations. Then, we present our approach for query transformation and propose necessary reformulation rules. In addition, the mappings that are needed to connect an ontology to its underlying database are discussed in Section 4. Finally, Section 5 concludes the paper.

2. Background

2.1. Ontology

Definition. The term *"Ontology"* or *"ontologies"* is becoming frequently used in many contexts of database and artificial intelligence researches. There are many definitions of what an ontology is [6, 7]. An initial definition was given by Tom Gruber: *"an ontology is an explicit specification of a conceptualization*" [6]. Ontologies have been increasingly emerging because of the crucial role that they play: Ontologies provide a concise and unambiguous description of concepts and their relationships for a domain of interest. This knowledge can be shared and reused by different participants.

Informally, we define an ontology as an intentional description of what is known about the essence of the entities in a particular domain of interest using abstractions, also called *concepts* and the *relationships* among them. Basically, the hierarchical organization of concepts through the inheritance ("`ISA`") relationship constitutes the backbone of an ontology. Other kinds of relationship like part-whole ("`PartOf`") or Synonym ("`SynOf`") or application specific relationships might also exist. Furthermore, a set of logical axioms is often associated with the ontology to specify semantics of the relationships. To the best of our knowledge, there is no work until now addressing the issue of using ontology relationships at the database instance level.

Formally, we define an ontology as finite sets ζ, $\Re$ and $\Im$ as follows: $\zeta = \{c_1, \ldots, c_n\}$ and $\Re = \{$"`ISA`","`SynOf`", "`PartOf`",$\ldots\}$, where $c_i \in \zeta$ is a concept name, $r_i \in \Re$ is a type of a binary relationship relating two concepts of ζ, and $\zeta \cap \Re = \emptyset$ (c_i and r_i are non-null strings). $\Im$ is a set of FOL [1]-expressions over $\Re$ describing dependencies between elements of $\Re$. Note that many real-world ontologies already combine data instances and concepts [7]. In our definition instances are not considered as part of an ontology.

Figure 1 depicts an example of a graph representation of a fragment of an ontology called "Entity Ontology" (denoted by O_1). This ontology describes a conceptualization of specific domains of real world entities. A part of this ontology is adopted from the ontology described in [17, 4].

Semantic of relationships. We assume that each of the following relationships: "`ISA`", "`SynOf`" and "`PartOf`" has the common semantics known in most information modelling systems. Further, ontological relationships can be interpreted using the axiom set $\Im$. We use the FOL-language to formulate axioms because of its great expressiveness. We interpret the "`ISA-`" and "`SynOf`$-$" relationships as transitive. However, for the "`PartOf`" relationship Transitivity depends on the context of the ontologies [2]. Transitivity can be expressed by the following axioms:

$\forall x\ y\ z\ ISA(x,y) \wedge ISA(y,z) \rightarrow ISA(x,z)$
$\forall x\ y\ z\ SynOf(x,y) \wedge SynOf(y,z) \rightarrow SynOf(x,z)$

Moreover, semantics of the other relationships which are dependent on the domain of the ontology need always to be specified. Most of these relationships are expressed in terms of each others as we shall see in Section 3.2.

1 First order logic (FOL)

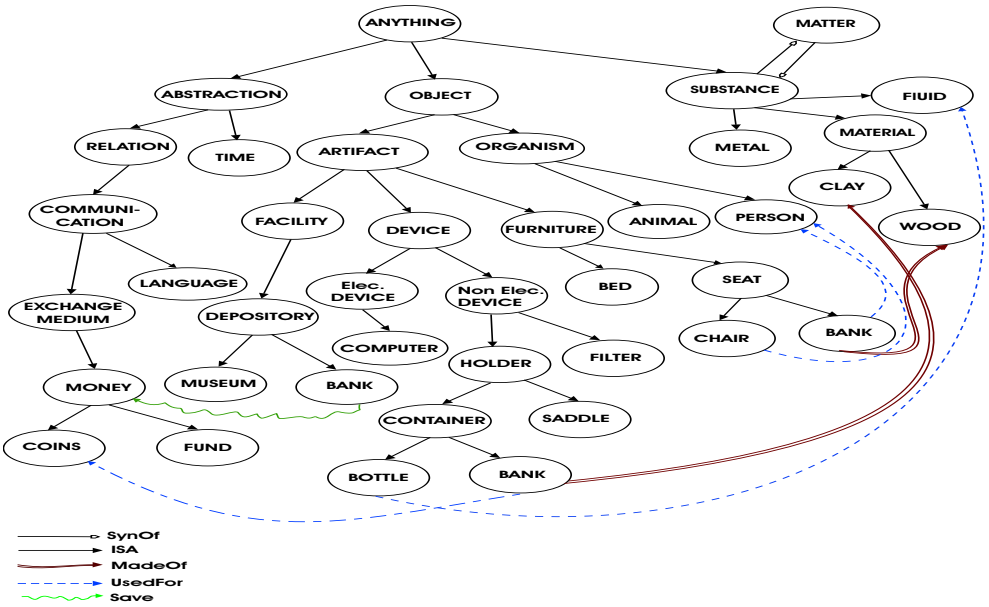

Figure 1. Entity Ontology O_1

2.2. Ontology Representation

An ontology can be represented as a directed labelled graph $G(V, E)$, where V is a finite set of vertices and E is a finite set of edges: Each vertex of V is labelled by a concept from ζ and each edge of E is labelled by an inter-concept relationship from $\Re$. Further, we refer to a node by its label (a concept) and refer to an edge by its two node concepts and its label (a relationship). We refer by $e = c_1\, R_i\, c_2$ the edge between two concept nodes c_1 and c_2 which is labelled by the relationship R_i of $\Re$. Mathematically, the ontology graph G can be represented by a relation $G \subseteq \zeta \times \zeta$, which is equal to the union of each relation R_i. Hence, an expression of the form $(c_1, c_2) \in R_i$ refers to $c_1\, R_i\, c_2$.

Alternatively, an ontology can be represented using RDF [2]-based models [10]. RDF is a standard language for describing and interchanging meta data. RDF is simply a series of statements of a triple-form *(Subject, Predicate,Object)*. The meaning of a particular RDF model is the conjunction (ie. logical AND) of all the statements it contains [10]. Any element of a statement is referred to as a *resource*. Predicates specify characteristics or relationships of resources, and are referred to as *properties*. In particular, RDF properties may be seen as attributes of resources in the context of databases. Therefore, we can simple match the ontology graph with the RDF model: Each edge in G could be encoded by an RDF statement, where the edge-nodes (concepts) correspond to the subject and the object, and the edge-label (relationship) corresponds to the predicate. However, a part of the ontology, namely the set $\Im$, could not be represented in RDF. RDF does provide no mechanisms for specifying the meanings of relationships nor any mechanisms for describing relationships between them. To overcome these shortcomings RDF has been extended to other models such as RDF Schema and OWL [3] [15].

In summary, an ontology is defined as the following set $O = \{G, \zeta, \Re, \Im\}$.

3. Ontology based Query Transformation

The objective of our approach to query processing is improving the query processing capabilities of DBMSs based on semantics derived from ontologies. The approach can be applied to the existing DBMSs in a simple manner without any complex modifications of their components. Figure 2 shows an overview on the system's architecture. The system mainly consists of three components. The first component is the transformation engine which constitutes the core of the system. It performs a pre-processing of a user query, say Q, before submitting it to the database. This is done by parsing Q and reformulating it into another query, say Q', using a set of semantic rules.

These rules rely on additional semantics extracted from an ontology. Basically, they must contribute to:

- Expand user queries by changing their select conditions using synonyms for the terms in the condition and others specifying them.

2 Resource Description Framework (RDF)

3 Ontology Web Language(OWL)

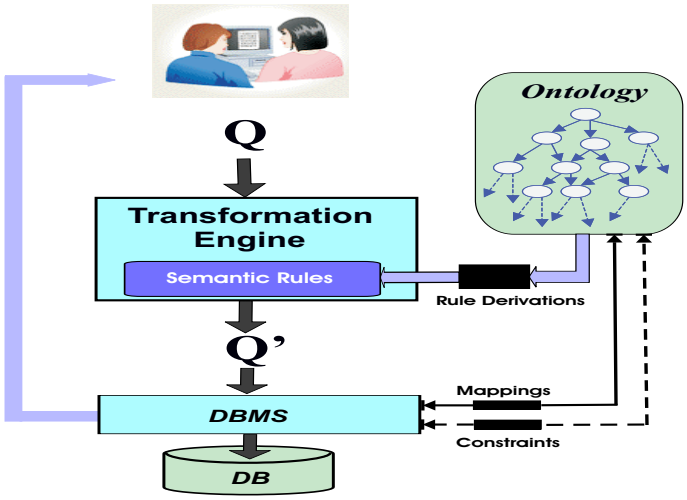

Figure 2. System Architecture

- Substitute the query conditions with other conditions that are semantically equivalent. Two sets of conditions are semantically equivalent if their corresponding concepts together with their related relationships identify the same concept w.r.t the associated ontology.
- Reduce the scope of queries by restricting its context.

During query transformation semantic rules are applied uniformly, in any order. This is done iteratively such that at each iteration the transformed query obtained in the previous iteration is used to generate another query until no more transformations are possible. It is possible that no rules can be applied to the query and the output query is then equal to the input query. When transformations happen and additional tuples are generated, the transformation engine allows users to distinguish them from the original query tuples by adding a label. The rule derivation process is done manually by ontology and database experts. We have developed a set of such rules based on information mappings between ontological and database elements. The second component is an ontology which is associated with the underlying database. It could be either a general or a domain-oriented ontology depending on the nature of the database in question. Here, the role of the ontology is to provide semantic knowledge about the data in the database. The ontology should not be build from scratch but existing ontologies could be reused. The dashed arrows represent a set of constraints and mappings between ontology and database elements that should be established for applying the transformation rules. In Section 4 we specify the properties of different kinds of mappings. The third component is the DBMS which processes the output query and returns the answer to the user. As results, the answer might contain more or less tuples than that answer expected by the original query. According to this feature we classify the reformulation rules into two categories: *Augmentation rules* and *reduction rules*. In this paper, we describe only one rule For the first category and only two rules for the second category; please refer to [16] for additional rules. The syntax of the presented rules are given in Appendix B.

3.1. Reduction Rules

The main feature of these rules is that after reformulating a user query the number of tuples in the answer might decrease compared to that number of tuples before any reformulation.

In the following, we describe one of such rules. We call this rule *"the sensitivity rule"* because its goal is to increase the *sensitivity* of a user query. A query is called *sensitive* if its answer contains as few as possible *false positives*. We define a tuple as *false positive* if it is semantically not correct w.r.t. the user's expectations.

For example, a problem might occur when querying a database containing homonymous terms. If a user queries a database using terms in his query expression that might be homonymous with some other terms in that database, the answer to his query might contain tuples that are irrelevant to him. For instance, the term "bank" has different meanings. It means either a container for keeping coins or a piece of furniture for sitting on or a financial institution for saving money [4]. Therefore, if a user queries a given database for information concerning an object "bank", the database might return tuples containing data about furniture or containers or institutions of type bank. This might not meet the user's intention if the user expects data only on furniture.

To solve this problem, we propose a reformulation rule based on the use of an ontology associated with the given

database. By applying this rule a context could be specified for a user query. That is, the context defined by the semantic description of the data, which uses vocabularies from the ontology to express the user's intention. The intuition is to specify user queries sufficiently to derive the relevant meaning based on the ontology concepts. Thus, in the example above, the user's intention to find information about "bank" as furniture can be specified by domain specific ontologies which can describe different aspects of furniture. That is, the context of user queries will be restricted to furniture.

However, if the ontology is more general i.e. specifies more than one context (e.g. O_1). In this case it would be difficult to determine the user's intention immediately. For example, the concept BANK might label two different nodes in two different subgraphs of the ontology. Each subgraph represents the related context of "bank". We suggest that the system asks the user to specify a unique "context". This could be done by providing him with the possibility to choose one of the contexts specified in the ontology in terms of the *immediate uncommon concepts* of the BANK concepts. The immediate uncommon concepts of two given concepts are defined in terms of the *least common concept* as follows:

Definition 1. Let a, b, l be concepts of ζ.
l is a *least common concept* (lcc) of a and b iff

- $a \in DESC("ISA", l)$ and $b \in DESC("ISA", l)$,
- $\forall k, k' \in \zeta$, if $a, b \in DESC("ISA", k) \cap DESC("ISA", k')$ then $k = k'$
- if $\exists\, c' \in \zeta \mid a \in DESC("ISA", c')$ and $b \in DESC("ISA", c')$ then $l \in DESC("ISA", c')$ □

Definition 2. Let a, b, m, m', g, and g' be concepts of ζ.
m and m' ($m \neq m'$) are *immediate uncommon concepts* (imuc) of a and b resp. iff

- $\exists\, l \in \zeta \mid l = lcc(a, b)$ AND
- $m = RChild("ISA", l) \wedge m' = RChild("ISA", l)$

For example, the immediate uncommon concepts of the concepts BOTTLE and CHAIR are the concepts DEVICE and FURNITURE, respectively, since their least common concept is the concept ARTIFACT. □

Next, we illustrate the sensitivity rule and its effectiveness by means of an example. A formal description of the rule is given in Appendix B.

Example 1. We assume a database DB_1 containing information about items of products. The DB_1 schema might have a relation, called 'Item', whose schema defines the name of each object, the material it is made of, its use and its price. An instance of DB_1 and a description of the relation 'Item' are given as follows:

ITEM(A-ID, Name, Made, Use, Price)
A-ID: Item identifier
Name: Item name
Made: Material type
Use: Purpose of use
Price: Item price
Primary-Key(A-ID)

A-ID	Name	Made	Use	Price
41	bed	wood	kid	120 $
42	bank	wood	person	300 $
43	chair	wood	person	150 $
44	flat iron	substance	clothes	60 $
45	chain	gold	women	850 $
46	perfume	roses	women	85 $
47	bank	clay	coins	50 $
48	cage	metal	birds	300 $

Table 1. Item relation

In addition, we assume the ontology O_1 is associated with DB_1. Note that O_1 contains additional domain relationships: "MadeOf", "UseFor" and "Save". The meaning of the "UseFor"-relationship, for example, is that if A (a concept) relates to B (a concept) by this relationship, the objects referred to A are used for purposes given by the objects referred to B.

Now, suppose that the user wants to retrieve all tuples from DB_1 concerning the container 'bank'. His query can be represented as following:

$$Q_1 = \{(x_1, x_2, x_3, x_4, x_5) \mid (x_1, x_2, x_3, x_4, x_5) \in ITEM \wedge x_2 = "bank"\}.$$

Obviously, the answer from the current DB_1 database to the query Q_1 contains the tuples 42 and 47. However, the tuple 42 does not meet the intention of the user since it relates to furniture. By using the ontology O_1 the system could deduce that "bank" is related to three different contexts: Furniture, device and facility. This is done by retrieving the *immediate uncommon concepts* of BANK concepts. Therefore, it has to ask again the user for specifying his query providing him the three relevant variants. If the user means a device "bank", the system will be able to specify the concept BANK from O_1 that the related objects are used for keeping coins. Thus, the user query should include terms which are related to the concept COINS to assert the intended context of the answer. The application of this rule to the query Q_1 leads to the following query:

$$Q_1' = \{(x_1, x_2, x_3, x_4, x_5) \mid (x_1, x_2, x_3, x_4, x_5) \in ITEM \wedge (x_2 = "bank" \wedge x_4 = "coins")\}.$$

Consequently, the answer to this reformulated query will contain only the tuple 47 as expected by the user.

3.2. Augmentation Rules

The goal of these rules is to extend the query answer with results that meet user's expectations. To this end, we have developed the following rules: a *Support-*, a *Feature*, a *Part-Whole* a *Vocabulary-* rule, and *Consistency rules* [16]. Briefly, the Support rule is based on semantics of the relationships from which the ontology is constituted. The Feature rule is based on domain-specific relationships that are mapped to the database model. The Part-Whole rule is based on the use of the "part-whole" properties to discover new database objects which are closely related to those the given query returns. In the following we present the Vocabulary- and Consistency rules in detail.

3.2.1. The Vocabulary Rule This rule addresses the problem of semantic ambiguities. Basically, the rule is developed to bridge the semantic gap of the vocabularies used in the user's query and those stored in the database. Intuitively, the rule reformulates the query condition by using semantically equivalent conditions based on terms derived from a given ontology. This is done by following synonym- and subsumption relationships ("`SynOf`" and "`ISA`"). A formal description of the rule is given in Appendix B.

Example 2. A user might submit a query to the database DB_1 as the follows:

$$Q_2 = \{(x_1, x_2, x_3, x_4, x_5) | (x_1, x_2, x_3, x_4, x_5) \in ITEM \wedge x_2 = "seat"\}.$$

According to the ontology O_1 the concept `SEAT` subsumes the concepts `CHAIR` and `BANK`. Intuitively, the `ISA`-relationship implies a strong similarity between a concept and its sub-concepts. Therefore, a seat could be also a chair or bank. By applying the rule we get the following new query:

$$Q_2^{'} = \{(x_1, x_2, x_3, x_4, x_5) | (x_1, x_2, x_3, x_4, x_5) \in ITEM \wedge (x_2 = "seat" \vee x_2 = "chair" \vee x_2 = "bank")\}.$$

Obviously, after applying the sensitivity-rule the transformation engine reformulates $Q_2^{'}$ again into another query that returns the tuples 42 and 43. Compared to the original query Q_2, which does not return any result, we conclude that the transformed query is more meaningful than Q_2 (returns "more complete" answer according to the semantic knowledge) .

3.2.2. The Consistency Rules The basic idea behind these rules is the use of relationships of certain concepts related to the query in order to derive other concepts, which have the same semantics, for transforming the query. To illustrate this idea we use an ontology O_2, called "Family Ontology", which describes concepts representing members of a family and the relationships among them. Figure 3 provides the graph representation of O_2. The logical interpretation of O_2 is presented in Appendix D.

Query Aspect. Basically, this rule has been developed for queries which have simple forms like queries that correspond to the Select-Project-Join(SPJ)-queries of the SQL-language. A query can be characterized by the following components:

- a set of attributes A_Q
- a set of relation names R_Q
- a set of conditions (constraints) I_Q

We call all these components the *query aspect* of Q.
A_Q contains the output attributes corresponding to each value in a result's tuple. A_Q matches with the select list of an SQL-query. R_Q contains the basic relations used in the query. I_Q are presented by predicates of the form: $u\theta v$ where u and v are constants or variables and θ is an arithmetic comparison operator (=, $<$ and so on). If u and v are both variables we refer to the corresponding condition as *join-condition*. If u is a variable and v is a constant we refer to the corresponding condition as *selection-condition.*

Atomic vs. derived Concepts/Relationships. Given an ontology, we distinguish between two kinds of concepts and relationships. A concept (or a relationship) is said to be *atomic* if it is not defined in terms of other concepts or relationships, otherwise, it is said to be *derived*. That is, if concepts and relationships are defined by axioms, for each derived concept (or relationship) say H, there is at least one axiom whose left-hand side is only H. Furthermore, atomic concepts (or relationships) do occur only on the right-hand side of axioms. We refer to concepts or relationships appearing in the right-hand side in an axiom of a derived concept or (a relationship) as being *constituents*. For instance, the concept `FATHER` in the family-ontology O_2 is a defined concept since it is defined in terms of the concepts `PARENT` and `MAN` according to the axiom $A2$ (see Appendix D). However, the concepts `WOMAN` and `MAN` are atomic (the only ones) since they do not appear in a left-hand side of any axiom. On the other hand, according to the axiom $F9$ `HasUncle`-relationship is a derived relationship since it is expressed in terms of `HasParent` and `HasBrother` relationships. However, the only atomic relationship of O_2 is the `HasChild`-relationship.

Now, from a database point of view, if a database represents instances of an atomic concept (or a relationship), this concept (or relationship) will be mapped to a base relation while if it represents instances of a derived concept (or relationship), this concept (or relationship) will be mapped to derived relations.

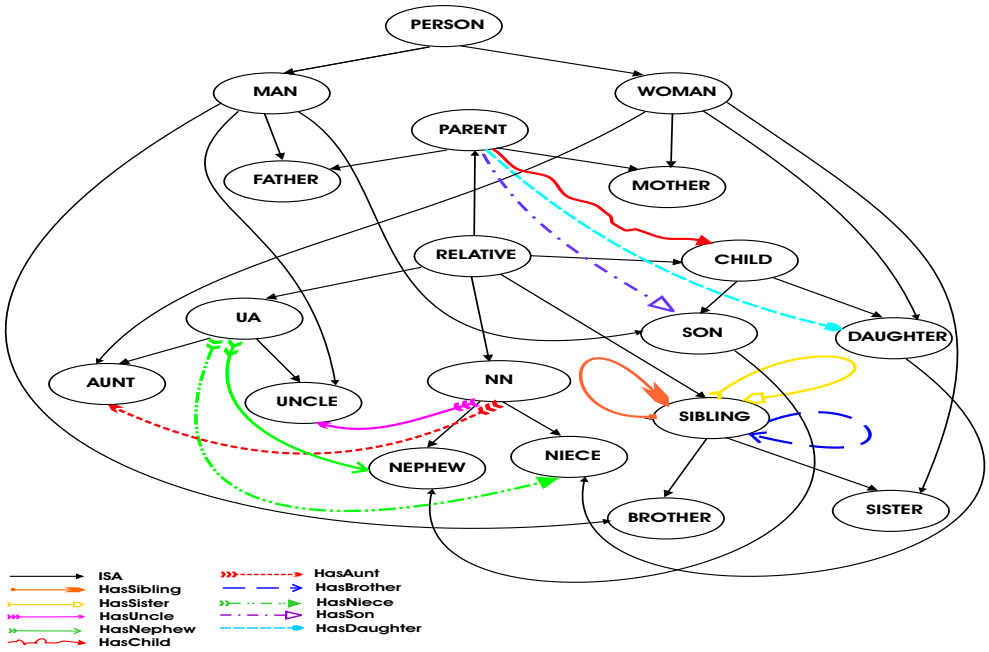

Figure 3. Family Ontology O_2

Ontology Consistency. a database is said to be *consistent* with respect to an ontology if the database instances representing ontology concepts satisfy all the axioms $\Im$ of the ontology. Moreover, if two axioms are equivalent then they must represent the same database instances. We define two axioms to be equivalent if they define the same derived concept or relationship (see example 3).

Query Semantics. The execution of a query over a database extension returns a set of tuples. Usually, the tuples share common semantics by describing common features of the objects they represent. These features can be expressed by a set of conditions which should be satisfied by these objects. For instance, assume we have a database, denoted by DB_2, which contains information about individuals of a family. Due to limited space we describe only the schema of DB_2 in Appendix C. Now suppose a user intends to request DB_2 to get information about persons and their nephews. His query might look like

$$Q_3 = \{(x_1, z_1) \mid x_1 \in Nephew \wedge \exists y_1, y_2\ [(y_1, y_2) \in NephewOf \wedge \exists\ z_1[z_1 \in Uncle \wedge x_1 = y_1 \wedge y_2 = z_1\ ]]\}$$

The evaluation of Q_3 against DB_2-database will return instances that represent the *cousinship* (relationship of cousins) between uncles and nephews. In this context, semantics of a query could be specified using a set of conditions over a set of attributes of the relations used in that query. These semantics can be defined using an ontology associated with the database to which the query is posed. In fact, at the ontology level, query semantics might be captured by a fragment of the ontology i.e. a set of particular concepts, the relationships that relate them, and their meanings (expressed by logical axioms). Thus, such a fragment represents also components of the query aspect in the ontology.
Formally, given a query Q, we define the *semantic vector* of Q, denoted ϖ_Q, as an ontology fragment described by the triple: $\varpi_Q = \langle \zeta_Q, \Re_Q, \Im_Q \rangle$ where $\zeta_Q \subset \zeta$, $\Re_Q \subset \Re$ and $\Im_Q \subset \Im$.

To determine ϖ_Q we need to map the components of the query aspect of Q to ontology elements. If the mapping is complete i.e. all components are mapped, then we say that ϖ_Q completely covers the semantics of the query, otherwise, ϖ_Q partially covers its semantics. We often encounter the latter situation if I_Q contains selection-conditions. For instance, the semantic vector of Q_3 is the following:

$$\varpi_{Q_3} = \langle \{UNCLE, NEPHEW\}, \{HasNephew\}, \{\forall x\ \ y \exists a\ HasNephew(x, y) \rightarrow HasSibling(x, a) \wedge HasSon(a, y)\}\rangle$$

We note that ϖ_{Q_3} completely covers the semantics of Q_3. However, for example, if a user specifies nephews to whom the uncle should be related then ϖ_{Q_3} will partially cover the semantics of the query because there are no mappings of database values in O_2. For instance, such a query might be the following:

$Q_4 = \{(x_1) \mid x_1 \in Nephew \wedge \exists y_1, y_2\ [(y_1, y_2) \in NephewOf \wedge \exists\ z_1[z_1 \in Uncle \wedge x_1 = y_1 \wedge y_2 = z_1 \wedge z_1 = 3]]\}$

We also note that if the selection-condition changes, the semantic vector does not change since ontologies do not contain instances (w.r.t. our definition). Therefore, the same fragment of O might cover multiple queries (i.e. mapped to multiple query aspects).

On the other hand, a given query might be covered by more than one fragment from the ontology representing equivalent semantics i.e. to a given query aspect we might assign more than one semantic vector. We define a set of semantics to be equivalent if they represent the same concept instances. One of the major reasons for this multiple representations is that concept instances, to which tuple values are mapped, could belong to several concepts and might play several roles (by participating in different relationships). Thus, another equivalent semantics of a query could be derived from the ontology based on the initial semantics representation. For example, an individual, say "John", could be an individual of the concept `NEPHEW` and at the same time an individual of the concept `SON`. If so, then "John" has to participate in the relationships `"HasUncle"` or `"HasAunt"` and `"HasSon"` i.e. he has an uncle, say "Smith", (or an aunt) and has a parent, say "Mark".

In order to derive other equivalent semantics, each relationship invoked by the query must be investigated using information from the set $\Im$ of O. If a relationship is a derived relationship, it could be then interpreted in terms of its constituents. Combined together, constituent-relationships allow us to capture additional meanings about a derived relationship. For example, the $\Im$ of the family-ontology O_2 contains the following axiom:

$$\forall x\ y\ \exists a HasNephew(x, y) \rightarrow HasSibling(x, a) \wedge HasSon(a, y) \quad (1)$$

This axiom indicates that if an individual x has a nephew y then it has a sibling whose son is y. According to this axiom the relationship `"HasNephew"` is specified in terms of the `"HasSibling"`- and `"HasSon"` relationships. Therefore, "Smith", the uncle of "John", has a sibling who might be the parent "Mark" whose son is "John". Consequently, determining the nephews of an individual of `UNCLE` has the same meaning as determining the sons of his siblings. Therefore, all concepts and logical axioms related to constituent-relationships constitute the content of an additional semantic vector. For instance, another semantics of Q_3 could be derived from O_2 as defined by the following semantic vector:

$\varpi'_{Q_3} = \langle \{SIBLING, PARENT, SON\},$
$\{HasSibling, HasSon\},$
$\{\forall x\ y\ HasSibling(x, y) \rightarrow HasSibling(y, x);$
$\forall x\ y\ HasSon(x, y) \rightarrow HasChild(x, y) \wedge SON(y)\}\rangle$

In summary, mapping a given query (i.e. query aspect) to its semantics in the ontology (i.e. semantic vector) might allow us to derive another equivalent semantics i.e. other ontology fragments that represent the same concept instances of its initial query semantics.

Mapping Specifications. So far we have described how multiple semantics of a query can be extracted from an ontology. Now, we want to illustrate how to specify their corresponding instances in the database. For this purpose we use mapping information about derived relationships: We define for each relationship new mapping using the existing mappings of its constituents. This information can be provided by the set $\Im$ of O as illustrated by the following example:

Given the axiom (1), according to the mapping Ψ_{rR} between O_2 and DB_2 `HasSibling`- and `HasSon`-relationships are mapped to the relations expressed by $\{(x) | (x) \in SiblingOf\}$ and $\{(x, y) \mid (x, y) \in Parenthood \wedge \exists z\ [(z) \in Son \wedge y = z]\}$, respectively. The result of a join between these relations corresponds to the database instances to which the relationship `"HasNephew"` can be mapped. It can be expressed by $\{(x, y) \mid (x, y) \in SiblingOf \wedge \exists z, t\ [(z, t) \in Parenthood \wedge \exists u, w[(u) \in Son \wedge y = z \wedge t = u]\}$.

Consequently, database instances, to which a derived relationship is mapped, could be determined by joining the relations corresponding to their constituents.
Let Δ_v be a set of expressions which define derived relations for each derived relationship of O. Formally, we define the new mapping as a relation ϕ over the sets $\Re$ and Δ_v. For example, Table 2 depicts this mapping type concerning the database DB_2 and its associated ontology O_2.

Relationship	**Derived relation**
HasSibling	$\{(x, y) \mid \exists p\ (p, x) \in Parenthood \wedge (p, y) \in Parenthood\}$
HasNephew	$\{(x, y) \mid (x, y) \in SiblingOf \wedge \exists z, t\ [(z, t) \in Parenthood \wedge \exists u[(u) \in Son \wedge y = z \wedge t = u]]\}$
HasNiece	$\{(x, y) \mid (x, y) \in SiblingOf \wedge \exists z, t\ [(z, t) \in Parenthood \wedge \exists u, w[(u) \in Daugther \wedge y = z \wedge t = u]]\}$
HasUncle	$\{(x, y) \mid (x, y) \in Parenthood \wedge \exists z, t\ [(z, t) \in SiblingOf \wedge \exists u[(u) \in Brother \wedge y = z \wedge t = u]]\}$
HasAunt	$\{(x, y) \mid (x, y) \in Parenthood \wedge \exists z, t\ [(z, t) \in SiblingOf \wedge \exists u[(u) \in Sister \wedge y = z \wedge t = u]]\}$

Table 2. Mapping ϕ for DB_2 and O_2

Consistency Rules Definition. Given a query Q, this rule aims at capturing additional semantics from the ontology, which are equivalent to the semantics of Q, in order to reformulate Q into another query Q'. The query Q' could provide more information relevant to users and additional results that meet his intention. The rule is based on the ideas discussed earlier. Intuitively, the rule is developed for the cases when there exists a mapping between relationships of the ontology and relation names that appear in Q. More clearly, if Q invokes a relation or a join between a set of relations which expresses a semantic relationship between concepts representing these relations in the ontology then Q can be formulated using a derived relation that corresponds to that relationship. The derived relation can be determined using additional mappings for that relationship. Consequently, after the reformulation, if a database is not consistent with its associated ontology, the answer of Q' might contain more tuples than the answer of Q. The additional tuples would interpret the semantics of the original query.

We have to distinguish two kinds of rules. The first rule (Rule 6a) concerns the case when there exists a mapping between a relationship of the ontology and a relation name invoked in the query mappings between concepts supporting that relationship and other relation names. The second rule (Rule 6b) concerns the case when there exists a mapping between only one relation in Q and a relationship in the ontology.

Example 3. Assuming that DB_2 includes the instances described below. Supposing that a user inquires DB_2 for nephews whose uncle has id=1. The submitted query may look like:

$Q_5 = \{x_1 \mid x_1 \in Nephew \wedge \exists y_1, y_2\ [(y_1, y_2) \in NephewOf \wedge \exists\ z_1[z_1 \in Uncle \wedge x_1 = y_1 \wedge y_2 = z_1 \wedge z_1 = 1\ ]]\}$

Obviously, the result of query Q_5 is the tuple 1.

NEPid
1
2

NEPid	UAid
1	1
1	2
2	3

Uid
1
2
3

Kid
1
2
3
4

Eid	Cid
2	1
4	3
4	1
5	4
6	2

Gid1	Gid2
1	4
1	5
2	4
3	6

Table 3. Nephew, NephewOf, Uncle, Son, Parenthood, and SiblingOf relations

Now, by applying the consistency rule (Rule 6a) to query Q_5 we obtain the following transformed query:

$Q'_5 = \{(s) |\ (s, g) \in V_0 \wedge \exists z_1 \in Uncle \wedge z_1 = g \wedge z_1 = 1]]\}$
where $V_0 = \{(g, s) \mid \exists (g, g') \in SiblingOf \wedge \exists e, c$
$[(e, c) \in Parenthood \wedge [s \in Son \wedge g' = e \wedge c = s\ ]]\}$.

The answer to Q'_5-query contains the tuples 1, 3 and 4. As a result, the user receives additional tuples which meet his intention because the sons identified by 3 and 4 are nephews of the mentioned uncle according to the ontology semantics provided by O_2. Thus, in this state the DB_2-database is not consistent with respect to the O_2-ontology.

4. Mappings between Ontologies and Databases

In order to accomplish the query reformulation task, mapping information between the ontology and the underlying database must exist.

This information links concepts and relationships of an ontology with instances of the underlying database. More precisely, there are three types of mappings:

1. mapping between concepts and attribute values,
2. mapping between concepts and relations, and
3. mapping between relationships and relations

In the following we illustrate the necessary properties of each type of mappings. For creating the mappings semi-automatic methods can be applied [9].

Let $O = \{G, \zeta, \Re, \Im\}$ be an ontology and DB be a database which has a schema $S = \{R_1, \ldots, R_n\}$ and an extension $Ext(DB)$. D is the set of all attribute domains in DB.

4.1. Mapping between concepts and attribute values

We define a simple many-to-many mapping Ψ_{CV} between concepts in ζ and values in the set D. The semantic of this mapping is that each value might be matched to a single or multiple concepts and a given concept might be matched more than one value. For example, the concept `BED`, in the ontology O_1 can be mapped to the value "bed" of the attribute `Name` in the relation `Item`. However, if a value has multiple homonyms, such as the term "bank", it might be mapped to multiple concepts.

Formally, we define Ψ_{CV} as a relation between ζ and D: $\Psi_{CV} \subseteq \zeta \times D$.

4.2. Mapping between concepts and relations

Definition 3: A *correspondence* ψ between an ontology concept $c \in \zeta$ and a set of database instances is defined as a pair of the form: $\psi =< c, Exp(R_1, \ldots R_k) >$ where $Exp(R_1 \ldots R_k)$ is a domain relational calculus expression. Exp defines a (derived) relation on $R_1, \ldots R_k$ containing only key values related to instances of a concept `C`. This kind of correspondence is called *concept correspondence*.

Types of correspondences: There are three types of concept correspondences:

1. correspondence between a concept and a (derived) relation,
2. correspondence between a concept and a set of relations, and
3. correspondence between a concept and a relation extension,

The first type of correspondence associates a given concept with a (derived) relation. More precisely, a correspondence could associate a concept with a set of tuples from a relation extension which satisfy certain conditions. For example, assuming that in DB_2 the attribute `Sex` of the relation `Person` takes values from the list {'male', 'female'}. Therefore, the concept `WOMAN` of O_2 representing women could be mapped to only a subset of the extension of `Person` that is related to tuples which have 'female' as value for the attribute `Sex`. Formally, this mapping element is defined as $< WOMAN, \{(x, y, z, w) \,|\, (x, y, z, w) \in Person \wedge w = "female"\} >$.

The second type of correspondence associates instances of a given concept with more than one relation. For instance, in O_2, the concept `CHILD` could be mapped to the relations Son and $Daughter$. Formally, this mapping element is defined as $< CHILD, \{(x)|(x) \in Son \vee (x) \in Daughter\} >$.

The third type of correspondence associates a concept with the whole extension of a relation. For instance, the concept `UNCLE` in the family-ontology might be mapped to the relation $Uncle$. This implies that each instance of the concept `Uncle` corresponds to one tuple of the relation $Uncle$. Formally, this mapping element is defined as $< UNCLE, \{(x)|(x) \in Uncle\} >$.

Definition 4: A *mapping* Ψ_{CR} between a set of ontological concepts ζ and a database extension $Ext(DB)$ is defined as a relation $\Psi_{CR} = \{\psi_i \,|\, \psi_i =< c_i, Exp(R_1, \ldots R_k) >\}$ where ψ_i is a concept correspondence.

From a semantic point of view the mapping Ψ_{CR} links each concept in the ontology to its corresponding tuples stored in the database.

4.3. Mapping between relationships and relations

Definition 5: A *correspondence* $\psi^{'}$ between an ontological relationship $r \in \Re$ and a set of database instances (tuples) is defined as a pair: $\psi^{'} =< r, Exp(R_1, \ldots R_k) >$, where $Exp(R_1 \ldots R_k)$ is a domain relational calculus expression. Exp defines a (derived) relation on $R_1, \ldots R_k$. Each element of this relation contains a pair of key values corresponding to concept instances that are related by r. This kind of correspondence is called *relationship correspondence*.

Types of correspondence: There are two types of relationship correspondences:

1. a correspondence between a relationship and a relation extension, and
2. a correspondence between a relationship and a (derived) relation

The first type of correspondence associates a relationship with all tuples of a relation extension. For instance, the relationship `HasChild` in O_2 relates the concept `PARENT` with the concept `CHILD`. This relationship could be mapped to the relation `Parenthood` implying that for each pair instances of `CHILD` and `PARENT` there exists a tuple in the relation `Parenthood` which contains only the corresponding key identifiers. Formally, this correspondence could be formulated as $< HasChild, \{(x, y) \mid (x, y) \in Parenthood\} >$.

The second type of correspondence associates a relationship with a set of tuples derived from more than one relation. For instance, the relationship `HasSon` in O_2 relates the concept `PARENT` with the concept `Son`. This relationship could be mapped to a set of tuples derived from the relations `Parenthood` and `Son`, namely to the result of the join between those relations. Formally, this correspondence could be formulated as $< HasSon, \{(x, y) \mid (x, y) \in Parenthood \wedge \exists z \,[(z) \in Son \wedge y = z]\} >$.

From a semantic point of view $\psi^{'}$ could be interpreted as the following: Given two concepts C_1 and C_2, if a relationship between C_1 and C_2 exists then each instance of C_1 could be related to an instance of C_2 through that relationship. We say that C_1 and C_2 support that relationship. For the concepts $C_1$and C_2 a mapping to the database must exist. The correspondence $\psi^{'}$ could be interpreted as the information that link related pairs of concept instances to pairs of key values in the database. Therefore, the mapping information of a relationship depends on the mapping information of each concept which it relates.

Definition 6: A *mapping* Ψ_{rR} between ontological relationships $\Re$ and a database extension $Ext(DB)$ is defined as a relation $\Psi_{rR} = \{\psi_i^{'} | < r_i, Exp(R_1, \ldots R_k) >\}$ where $\psi_i^{'}$ is a relationship correspondence.

For example, The mapping Ψ_{rR} between the ontology O_2 and its underlying database DB_2 is depicted in Appendix C.

5. Conclusion

Traditional approaches to query processing aim at rewriting a user query into another one which provides the same answer but is executed more efficiently. In this paper we address the problem of rewriting from another perspective. Our goal is to transform a user query into another query which is not necessary equivalent but can provide more meaningful answer. This answer might contains more or less tuples. In both cases it would meet the user's intention. To this end, we associated an ontology with a database and explained how it could be effectively exploited to capture semantics for the transformation process. We proposed a set of transformation rules and illustrated their effectiveness by some running examples. Although these rules might not be ideal, we hope that they can bring more insight into the nature of query answers. Furthermore, we specified the features of the mappings that are required to link the database with its associated ontology. Our approach could be implemented in current (object) relational database management systems without complex modifications of their main modules.

Currently, we are designing and implementing a prototype based on our approach. To this end, we attempt to reuse an existing ontology and adapt it to an associated database w.r.t. our framework. In the future work, we intend to extend the approach to allow the use of semantic rules in federated database systems.

References

[1] K. Aberer and G. Fischer. Semantic query optimization for methods in object-oriented database systems. In *IEEE International Conference Data Engineering*, pages 70–79, 1995.

[2] A. Artale, E. Franconi, N. Guarino, and L. Pazzi. Part-whole relations in object-centered systems: an overview, 1996.

[3] D. Beneventano, S. Bergamaschi, and C. Sartori. Description logics for semantic query optimization in object-oriented database systems. ACM Transaction on Database Syststems. Volume 28: 1-50, 2003.

[4] C. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[5] J. Grant, J. Gryz, , J. Minker, and L. Raschid. Semantic query optimization for object databases. ICDE, November 1997.

[6] T. Gruber. A translation approach to portable ontology specifications. In *Knowledge Acquisition, USA*, pages 199–220, 1993.

[7] N. Guarino and P. Giaretta. Ontologies and knowledge bases: towards a terminological clarification. In *Knowledge Building Knowledge Sharing,ION Press*, pages 25–32, 1995.

[8] J. W. Han, Y. Huang, N. Cercone, and Y. J. Fu. Intelligent query answering by knowledge discovery techniques. In *IEEE Trans*, pages 373–390, 1996.

[9] M. Hernandez, R. J. Miller, and L. M. Haas. Clio: A semi-automatic tool for schema mapping. In *ACM SIGMOD*, 2001.

[10] G. Klyne and J. Carroll. Resource description framework (rdf): W3c recommendation. availble at: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/, 2004.

[11] L. Lakshmanan and R. Missaoui. On semantic query optimization in deductive databases. In *IEEE Inter-national Conference on Data Engineering*, pages 368–375, 1992.

[12] A. J. Lee, A. Nica, and E. A. Rundensteiner. Keeping virtual information resources up and running. In *CASCON '97: Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*, page 13. IBM Press, 1997.

[13] A. Levy. Answering queries using views: A survey. *The VLDB Journal the International Journal on Very Large Data Bases*, 10:270–294, 2001.

[14] A. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 95–104, San Jose, Calif., 1995.

[15] D. L. McGuinness and F. Harmelen. Resource description framework (rdf): Concepts and abstract syntax. w3c recommendation. availble at: http://www.w3.org/TR/owl-features/, 2004.

[16] C. B. Necib and J. Freytag. Using Ontologies for Database Query Reformulation. In *Proc. on the 18 th conference on Advances in Databases and Information Systems*, 2004.

[17] A. Pease. The sigma ontology development environment. In *IJCAI-03 Workshop on Ontologies and Distributed Systems (ODS'03),Acapulco, Mexico*, 2003.

A. Graphical Operations

A set of primitive graph operations: $RParent$, $DESC$, $SUBT$ and $SYNs$, are defined as follows.

Let $P_{ths}(c_i - c_j)$ be a set of directed paths between two concept nodes c_i and c_j. Let be $c_1, c_2, s_k, s_h \in \zeta$ and $R, R_i \in \Re$:

- $SYNs(c_1) = \{c_2 \mid G(c_1, \text{"SynOf"}, c_2)\}$
- $Rparent(R, c_1) = \{c_2 \mid G(c_2, R, c_1)\}$
- $SUBT(c_1) = \{c_2 \in \zeta \mid \exists P_{ths}(c_1 - c_2)\}$
- $DESC(R, c_1) = \{c_2 \in \zeta \mid \exists p \in P_{ths}(c_1 - c_2) : \forall e = (s_k R_i s_h) \in p\ ,\ R_i = R\}$ □

Informally, $DESC(R, c)$ returns the set of all descendant concepts of c (a concept) following edges of type R. Similarly, $SUBT(c)$ returns all descendants of c for any edge-type and $SYNs(c)$ returns the set of all synonyms of c. $Rparent(R, c)$ returns concepts that correspond to ascendants of the node of c following edges of type R.

B. Syntax of Query Transformation Rules

Notations. Let U be a set of attributes $A_1, \ldots, A_n$ with domains $dom(A_i)$. Let D be the set of all attribute domains. A database schema consists of a set of relation schemas $\Sigma = \{R_1, \ldots, R_m\}$ with $R_i \subseteq U$. Furthermore, we choose the Domain relational calculus (DRC) to represent user queries. Given an element t_0 be element of D, the vocabulary rule, sensitivity rule and the view-based rules are formulated as follows.

B.1. Sensitivity Rule

IF $Q = \{x_i \mid (x_1, \ldots, x_n) \in R \wedge x_p \theta t_0\}$
and $\exists\, c_0, c_p \in \zeta \,|\, (c_0, t_0) \in \Psi_{CV}$ and $(c_p, A_p) \in \Psi_{CA}$
and $\exists A_i, \ldots, A_j \in U(R) \mid (r_k, A_k) \in \Psi_{rA}$
$r_k \in \Re \setminus \{"PartOf"\}$ and $(c_k, A_{pk}) \in \Psi_{CA}$
and $c_0 \in SUBT(c_k) \cap SUBT(c_p)$

THEN $Q' = \{(x_1, \ldots, x_n) | \, (x_1, \ldots, x_n) \in R \wedge (x_p \theta t_0) \bigwedge_{k=i}^{j} (\bigvee_{h=1}^{m} x_k \theta t_{kh})\}$

where $t_{kh} \in I_0 \cup I_1 \cup I_2$, $m = |I_0 \cup I_1 \cup I_2|$
$I_0 = \{t \in D \mid (c_{kh}, t) \in \Psi_{CV}\}$
$I_1 = \{t \in D \mid \exists s \in DESC("\texttt{ISA}", c_{kh}) \wedge (s, t) \in \Psi_{CV}\}$
$I_2 = \{t \in D | \exists s \in (SYNs(c_{kh}) \vee \in SYNs(a)) \wedge (s, t) \in \Psi_{CV}$, $a \in DESC("\texttt{ISA}", c_{kh})\}$
$c_{kh} \in DESC("ISA", c_k) \cap RParent(r_k, c_0)$,
$i =< k =< j =< n, k \neq p,\ 1 =< h =< m.$ □

B.2. Vocabulary Rule

IF $Q = \{(x_1, \ldots, x_n) \mid (x_1, \ldots, x_n) \in R \wedge x_i \theta t_0\}$
and $\exists c_0 \in \zeta,\ (c_0, t_0) \in \Psi_{CV}$

THEN $Q' = \{(x_1, \ldots, x_n) \mid (x_1, \ldots, x_n) \in R \wedge (\bigvee_{k=0}^{m} x_i \theta t_k)\}$

where $t_k \in I_0 \cup I_1, 0 \leq k \leq m = |I_0 \cup I_1|$
$I_0 = \{t \in D | \exists s \in (DESC('ISA', c_0)\} \cup \{(s, t) \in \Psi_{CV}\}$,
$I_1 = \{t \in D \mid \exists s \in SYNs(c) \wedge (s, t) \in \Psi_{CV}, c \in DESC('ISA', c_0)\}$. □

B.3. Consistency Rules

Rule 6a:

IF $Q = \{(x_1, \ldots, x_n) \mid (x_1, \ldots, x_n) \in R_1 \wedge \exists y_1, \ldots, y_n [\, (y_1, \ldots, y_n) \in R_2 \wedge \exists z_1, \ldots, z_n [\, (z_1, \ldots, z_n) \in R_3 \wedge x_1 = y_1 \wedge y_2 = z_1 \wedge z_p \theta t_0 \,]]\}$
and $\exists r_0 \in \Re | (r_0, R_2) \in \Psi_{rR}$
and $\exists c_1, c_2 \in \zeta \,|; (c_1, R_1) \in \Psi_{CR} \wedge (c_2, R_3) \in \Psi_{CR}$
and $\exists V_0 \in \Delta_v | (r_0, V_0) \in \phi$

THEN $Q' = \{(x_1, \ldots, x_n) \parallel (x_1, \ldots, x_n) \in R_1 \wedge \exists w_1, \ldots, w_n [\, (w_1, \ldots, w_n) \in V_0 \wedge \exists z_1, \ldots, z_n [\, (z_1, \ldots, z_n) \in R_3 \wedge x_1 = w_n \wedge w_1 = z_1 \wedge z_p \theta t_0 \,]]\}$. □

Rule 6b:

IF $Q = \{(x_1, \ldots, x_n) \mid (x_1, \ldots, x_n) \in R_1 \wedge \exists\, y_1, \ldots, y_n [(y_1, \ldots, y_n) \in R_2 \wedge x_2 = y_1 \wedge x_p \theta t_0]\}$
and $\exists r_0 \in \Re | (r_0, R_2) \in \Psi_{rR}$
and $\exists (x, y) \in \phi : x = r_0 \wedge y = V_0$
and $\exists V_0 \in \Delta_v | (r_0, V_0) \in \phi$

THEN $Q' = \{(w_1, \ldots, w_n) \mid (w_1, \ldots, w_n) \in V_0 \wedge \exists y_1, \ldots, y_n [\, (y_1, \ldots, y_n) \in R_2 \wedge w_2 = y_1 \wedge w_n = x_1 \wedge y_p \theta t_0 \,]]\}$. □

C. Family Database Schema

Person(Pid, Name, Address, Sex)
PKey(Pid)

Woman(Wid)
PKey(Wid)
FKey(Wid) references Person

Man(Mid)
PKey(Mid)
FKey(Mid) references Person

Parent(Eid)
PKey(Eid)
FKey(Eid) referencess to Person

Parenthood(Eid, Cid)
PKey(Eid, Cid)
FKey(Eid) references Parent
FKey(Cid) references Child

Father(Fid)
PKey(Fid)
FKey(Fid) references Man
FKey(Fid) references Parent

Mother(Mid)
PKey(Mid)
FKey(Mid) references Woman
FKey(Mid) references Parent

Child(Cid)
PKey(Cid)
FKey(Cid) references Person

Son(Kid)
PKey(Kid)
FKey(Kid) references Child
FKey(Kid) references Man

Daughter(Did)
PKey(Did)
FKey(Did) references Child
FKey(Did) references Woman

Sibling(Gid)
PKey(Gid)
Fkey(Gid) references Child

SiblingOf(Gid1,Gid2)
PKey(Gid1, Gid2)
Fkey(Gid1) references Sibling
Fkey(Gid2) references Sibling

Brother(Bid)
PKey(Bid)
FKey(Bid) references Man
FKey(Bid) references Sibling

Sister(Sid)
PKey(Sid)
FKey(Sid) references Woman
FKey(Sid) references Sibling

Uncle(Uid)
PKey(Uid)
FKey(Uid) references Man
FKey(Uid) references Sibling

Aunt(Aid)
PKey(Aid)
FKey(Aid) references Woman
FKey(Aid) references Sibling

Niece(Nid)
PKey(Nid)
FKey(Nid) references Daughter

NieceOf(Nid, UAid)
PKey(Nid, UAid)
FKey(Nid) references Niece

Nephew(NEPid)
PKey(NEPid)
FKey(NEPid) references Son

NephewOf(NEPid, UAid)
PKey(NEPid, UAid)
FKey(NEPid) references Nephew

FKey(*id-name*) and PKey(*id-name*) denote foreign key and primary key constraints respectively.

D. Family Ontology

$\forall x\ PERSON(x) \leftrightarrow MAN(x) \vee WOMAN(x)$ $(A1)$
$\forall x\ FATHER(x) \leftrightarrow PARENT(x) \wedge MAN(x)$ $(A2)$
$\forall x\ MOTHER(x) \leftrightarrow PARENT(x) \wedge WOMAN(x)$ $(A3)$
$\forall x\ SON(x) \leftrightarrow CHILD(x) \wedge MAN(x)$ $(A4)$
$\forall x\ DAUGHTER(x) \leftrightarrow CHILD(x) \wedge WOMAN(x)$ $(A5)$
$\forall x\ BROTHER(x) \leftrightarrow SIBLING(x) \wedge MAN(x)$ $(A6)$
$\forall x\ SISTER(x) \leftrightarrow SIBLING(x) \wedge WOMAN(x)$ $(A7)$
$\forall x\ UA(x) \leftrightarrow UNCLE(x) \vee AUNT(x)$ $(A8)$
$\forall x\ NN(x) \leftrightarrow NEPHEW(x) \vee NIECE(x)$ $(A9)$
$\forall x\ RELATIVE(x) \leftrightarrow UA(x) \vee NN(x) \vee SIBLING(x) \vee CHILD(x) \vee PARENT(x)$ $(A10)$
$\forall x \exists y PARENT(x) \rightarrow PERSON(x) \wedge HasChild(x,y)$ $(A11)$
$\forall x \exists p\ CHILD(x) \rightarrow PERSON(x) \wedge HasParent(x,p)$ $(A12)$
$\forall x \exists y SIBLING(x) \rightarrow PERSON(x) \wedge HasSibling(x,y)$ $(A13)$
$\forall x \exists y UNCLE(x) \rightarrow MAN(x) \wedge (HasNephew(x,y) \vee HasNiece(x,y))$ $(A14)$
$\forall x \exists y UNCLE(x) \rightarrow MAN(x) \wedge HasSibling(x,y) \wedge PARENT(y)$ $(A15)$
$\forall x \exists y AUNT(x) \rightarrow WOMAN(x) \wedge (HasNephew(x,y) \vee HasNiece(x,y))$ $(A16)$
$\forall x \exists y AUNT(x) \rightarrow WOMAN(x) \wedge HasSibling(x,y) \wedge PARENT(y)$ $(A17)$
$\forall x NIECE(x) \leftrightarrow DAUGHTER(x) \wedge (HasUncle(x,y) \vee HasAunt(x,y))$ $(A18)$
$\forall x NEPHEW(x) \leftrightarrow SON(x) \wedge (HasUncle(x,y) \vee HasAunt(x,y))$ $(A19)$
$\forall xy HasDaughter(x,y) \leftrightarrow HasChild(x,y) \wedge DAUGHTER(y)$ (F1)
$\forall xy\ HasSon(x,y) \leftrightarrow HasChild(x,y) \wedge SON(y)$ (F2)
$\forall xy\ HasSibling(x,y) \leftrightarrow HasSibling(y,x)$ (F3)
$\forall xyz\ HasSibling(x,y) \leftrightarrow HasSibling(x,z) \wedge HasSibling(z,y)$ (F4)
$\forall xy \exists p\ HasSibling(x,y) \rightarrow HasParent(x,p) \wedge HasParent(y,p)$ (F5)
$\forall xy\ HasParent(x,y) \leftrightarrow HasChild^{-1}(x,y)$ (F6)
$\forall xy HasSister(x,y) \leftrightarrow HasSibing(x,y) \wedge SISTER(y)$ (F7)
$\forall xy\ HasBrother(x,y) \leftrightarrow HasSibling(x,y) \wedge BROTHER(y)$ (F8)
$\forall xy \exists p\ HasUncle(x,y) \rightarrow HasParent(x,p) \wedge HasBrohter(p,y)$ (F9)
$\forall xy \exists p\ HasAunt(x,y) \rightarrow HasParent(x,p) \wedge HasSister(p,y)$ (F10)
$\forall xy \exists a\ HasNephew(x,y) \rightarrow HasSibling(x,a) \wedge HasSon(a,y)$ (F11)
$\forall xy \exists a\ HasNiece(x,y) \rightarrow HasSibling(x,a) \wedge HasDaughter(a,y)$ (F12)
$\forall xyz\ ISA(x,y) \wedge ISA(y,z) \rightarrow ISA(x,z)$ (F13)

Relationship	Database Instances
HasChild	$\{(x,y)\|(x,y) \in Parenthood\}$
HasParent	$\{(x,y)\|(x,y) \in Parenthood\}$
HasSibling	$\{(x,y)\|(x,y) \in SiblingOf\}$
HasNephew	$\{(x,y)\|(x,y) \in NephewOf\}$
HasSon	$\{(x,y) \mid (x,y) \in Parenthood \wedge \exists z\ [(z) \in Son \wedge y = z]\}$
HasDaughter	$\{(x,y) \mid (x,y) \in Parenthood \wedge \exists z\ [(z) \in Daughter \wedge y = z]\}$
HasSister	$\{(x,y) \mid (x,y) \in SiblingOf \wedge \exists z\ [(z) \in Sister \wedge y = z]\}$
HasBrother	$\{(x,y) \mid (x,y) \in SiblingOf \wedge \exists z\ [(z) \in Brother \wedge y = z]\}$
HasUncle	$\{(x,y) \mid [(x,y) \in NephewOf \wedge \exists z\ [(z) \in Uncle \wedge y = z]] \vee [(x,y) \in NieceOf \wedge \exists t[(t) \in Uncle \wedge w = t]]\}$
HasAunt	$\{(x,y) \mid [(x,y) \in NephewOf \wedge \exists z\ [(z) \in Aunt \wedge y = z]] \vee [(x,y) \in NieceOf \wedge \exists t[(t) \in Aunt \wedge w = t]]\}$

Table 4. Mapping between relationships and DB instances Ψ_{rR}