*KRDB Research Centre Technical Report:*

# Supporting the Design of Ontologies for Semantic Data Access

L. Lubyte, S. Tessaris

| | |
|---|---|
| **Affiliation** | KRDB Research Centre for Knowledge and Data, Free University of Bozen-Bolzano, Via della Mostra 4, 39100 Bolzano, Italy |
| **Corresponding author** | L. Lubyte lubyte@inf.unibz.it |
| **Keywords** | ontology design, predicate emptiness, ontology based data access |
| **Number** | KRDB09-03 |
| **Date** | 04-05-09 |
| **URL** | http://www.inf.unibz.it/krdb/ |

# Abstract

The use of ontologies to mediate the access and integration of relational data sources has been proved successful by recent works in the area of the Semantic Web. In particular, we consider a common information integration scenario in which several relational sources can be queried through a single (global) ontology providing a reconciled and integrated view of the underlying sources. Although data sources can be integrated into the information system by providing direct mappings from the relational tables to the given ontology, we believe that a more principled approach consists in wrapping each information source by means of a (local) ontology which precisely characterise the data source. The process of mapping the local ontology to the global one is facilitated by the fact that global and local ontologies share the same logical formalism.

While the global ontology provides a general view of the whole system, the local ontologies must be close to the data they wrap. This means that the new terms that they introduce must be "supported" by data from the relational sources; i.e. when queried, they should return nonempty answers. The problem of designing these wrapping ontologies is non-trivial and to tackle it those wrappers are usually carefully handcrafted by taking into account the query answering mechanism. In this report we address the problem of supporting the ontology engineer in this task. We provide an algorithm for verifying emptiness of a term in the enriched ontology w.r.t. the data sources. In addition, we show how this algorithm can be used to guide the ontology engineer in fixing potential terms unsupported by the data.[1]

[1]Preliminary results on the research reported here appeared in the Working Notes of Description Logics Workshop [14].

# Contents

# Chapter 1

# Introduction

## 1.1 Motivations

The use of a conceptual model or an ontology to wrap and describe relational data sources has been shown to be very effective in several frameworks involving management and access of data. These include federated databases [19], data warehousing [6], information integration through mediated schemata [12], and the Semantic Web [10] (for a survey see [21]). Ontologies provide a conceptual view of the application domain, which is closer to the user perspective. The automated reasoning can be leveraged to provide a better user support in exploring and querying the underlying data sources.

In this report we focus on the problem of designing ontologies which describe a relational data source, and whose purpose is to provide a semantically enriched access to the underlying data. We use the term *data wrapping ontologies* to distinguish these ontologies from domain ontologies; whose purpose is to model a domain.

In order to illustrate the different roles of the two kinds of ontologies let us consider a scenario in which several independent databases containing data on a given domain (e.g. showbiz) should be accessed through a single web portal driven by an ontology. This ontology is tailored to provide a general view over the domain and enable the users (or software agents) to retrieve information from the portal. For this reason the "portal ontology" doesn't necessarily use the same vocabulary as the data sources and we consider it a domain ontology. In order to retrieve data from the sources, the domain ontology should be "connected" to the data sources. This can be done either by defining mappings from the relational tables in the data sources to the terms in the ontology (see e.g [18]), or by creating smaller ontologies wrapping the data sources and aligning those with the domain ontology.

We believe that the second approach has several advantages over the use of direct mappings over the data sources. First of all, it enhances the modularity of the system, since the different components can be designed independently and then integrated. In fact, the domain ontology can be designed almost independently of the details of the data sources; while data wrapping ontologies are smaller and can be automatically bootstrapped from the data sources (see [15]). Moreover, there are well established techniques to support the alignment of ontologies (see [9]).

In order to maximise the benefits of using data wrapping ontologies, these should be rich enough to ease their integration with the domain ontology and, at the same time, precisely characterise the data they wrap. Ontologies extracted automatically from data sources (e.g. by analysing the constraints in the logical schema) are faithful representations of the data sources; however, they are usually shallow and with a limited vocabulary. For this reason, they can be used as bootstrap ontologies, and the task of enriching the extracted ontology is crucial in order to build a truly effective ontology-based information access system. The process of enriching an ontology involves at least the introduction of new axioms and/or new terms. While, from a purely ontological viewpoint, an ontology can be arbitrarily modified, we need to bear in mind that the ultimate purpose of the data wrapper is to access the information available from the data sources. This means that we should be able to use the newly introduced terms in order to retrieve data from the sources.

It is easy to provide examples where such new terms can be completely useless; in the sense that queries over these terms will always return empty answers. This not necessarily because they are unsatisfiable in the usual model theoretic meaning but because there is no underlying data supporting them. Let us consider the simple example depicted in Figure 1.1, where the bottom part represents the logical schema, the middle part the data source terms (connected with the relational sources) and the top part the enriched fragment of the ontology. It is obvious that any query on the *Actor* would return an empty answer, whatever the data sources may contain; on the other end, the concept represented by the same term would be satisfiable. The situation would be different if the *Actor* was also restricted to elements whose range w.r.t. *person_role* is restricted to *ActingRole*. In this case, there can be instances of the database for which the same query on *Actor* would return a non empty answer.

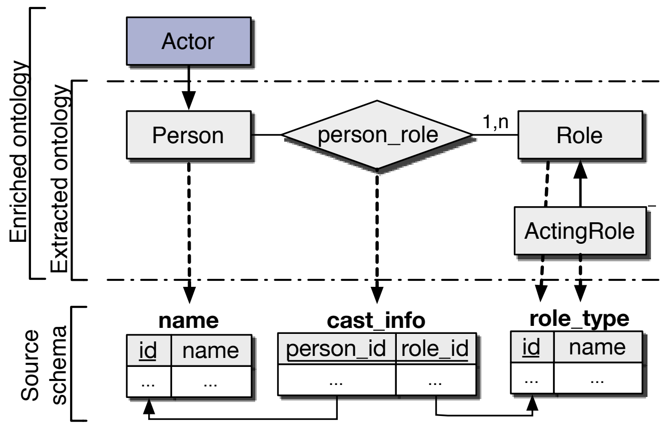


Figure 1.1: Example of simple data wrapper.

In order to ensure that queries over ontologies wrapping data sources provide sensible answers, these ontologies must be carefully handcrafted by taking into account the query answering algorithm. To the best of our knowledge, little or no research has been devoted to the support of the ontology engineer in such a complex and error prone task. Our research is directed to techniques and tools to support this modelling process.

The foundation of our technique is the problem of verifying the emptiness of a given term w.r.t. a set of data source terms (i.e. terms "connected" to data sources). Given a Description

Logic (DL) theory composed by Tbox and Abox over a given vocabulary (see Section 2.1 for details), we define a subset of the concepts and roles as *data source* terms. Given a Tbox, a concept or role term is empty iff the certain answer of the query defined by the term is empty for all possible Aboxes whose assertions are restricted to data source terms. The idea is that data (by means of Abox assertions) can only be associated to data source terms. Clearly the problem is different from classical (un)satisfiability, because we impose a restriction on the kind of allowed Abox assertions. Note that the two problems coincide when all the DL terms are considered as data sources.

In [14] we introduced the above problem and presented some preliminary results; while the contribution of this report is a generalisation the results presented in [14], by providing algorithms to verify term emptiness for a more expressive class of ontology languages (see [3]). In particular, a crucial gain in terms of expressive power of the language adopted in this work is the ability to express inclusions among roles. In addition, we describe how this algorithm can be used to support the user in the “repair” of the empty terms and we present a Protégé plug-in implementing it. Finally, we present an empirical study showing the benefits of our approach.

## 1.2 Outline of the Report

The report is structured as follows: in the next section we introduce the formal framework upon which our technique lays its foundations. Then, in Chapter 3 we present the emptiness testing algorithm and show how the information generated by this algorithm can be used to repair empty ontology terms. In Chapter 4 we present a Protégé plug-in implementing our approach, and a usability study involving external users. Then, in Chapter 5 we place our work within the related literature. Finally, in Chapter 6 we draw the conclusions and our future research plans.

# Chapter 2

# Formal Framework

In this chapter we introduce the formal framework for representing the ontology, queries over the ontology, and show how the actual ontology is linked to relational data sources. Then, we define the notion of predicate emptiness w.r.t. the data sources.

## 2.1 Ontology Language

The ontology language is based on the DL $\mathcal{ELHI}$ [3], an extension of $\mathcal{EL}$ [2], whose expressions are built over an alphabet comprising symbols for atomic concepts, roles and constants. We will consider constants in $\mathcal{ELHI}$ to be defined over the alphabet $\Gamma_O$ (disjoint from $\Gamma_V$ above) of constant symbols for objects. Then, the abstract syntax for $\mathcal{ELHI}$ concept expressions is the following:

$$B \rightarrow A \mid \exists R \mid \exists R^- \mid \exists R.A \mid \exists R^-.A \mid B_1 \sqcap B_2,$$

where $A$ is an *atomic concept*, $R$ is an *(atomic) role*, and $B$ is a *basic concept*. A *TBox* $\mathcal{T}$ is a set of *inclusion assertions* of the form $B_1 \sqsubseteq B_2$ or $R_1 \sqsubseteq R_2$, and an *ABox* $\mathcal{A}$ is a set of *membership assertions* of the form $A(a)$ or $R(a, b)$ with $a$ and $b$ constants in $\Gamma_O$. An $\mathcal{ELHI}$ *ontology* $\mathcal{K}$ is a pair $\langle \mathcal{T}, \mathcal{A} \rangle$, with $\mathcal{T}$ a TBox and $\mathcal{A}$ an ABox.

As usual, the semantics is given in terms of first-order interpretations. An *interpretation* is a pair $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ with $\Delta$ an interpretation domain of objects, and $^{\mathcal{I}}$ an *interpretation function* that maps each atomic concept $A$ to a subset $A^{\mathcal{I}} \subseteq \Delta$, each role $R$ to a binary relation $R^{\mathcal{I}} \subseteq \Delta \times \Delta$, and each constant $a$ in $\Gamma_O$ to an element $a^{\mathcal{I}} \in \Delta$. Moreover, for each $a, b \in \Gamma_O$, $a \neq b$ implies $a^{\mathcal{I}} \neq b^{\mathcal{I}}$, i.e., we adopt the *unique name assumption*. The interpretation function is extended to basic concepts as follows:

$$\begin{array}{ll} (\exists R)^{\mathcal{I}} = \{a \mid \exists b.(a, b) \in R^{\mathcal{I}}\} & (\exists R^-)^{\mathcal{I}} = \{a \mid \exists b.(b, a) \in R^{\mathcal{I}}\} \\ (\exists R.A)^{\mathcal{I}} = \{a \mid \exists b.(a, b) \in R^{\mathcal{I}} \rightarrow b \in A^{\mathcal{I}}\}^{1} & (B_1 \sqcap B_2)^{\mathcal{I}} = B_1^{\mathcal{I}} \cap B_2^{\mathcal{I}} \end{array}$$

An interpretation $\mathcal{I}$ *satisfies* an inclusion assertion $B_1 \sqsubseteq B_2$ (resp. $R_1 \sqsubseteq R_2$) if $B_1^{\mathcal{I}} \subseteq B_2^{\mathcal{I}}$ (resp. $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$). $\mathcal{I}$ satisfies a membership assertion $A(a)$ (resp. $R(a, b)$) if $a \in A^{\mathcal{I}}$ (resp. $(a, b) \in R^{\mathcal{I}}$). Finally, $\mathcal{I}$ is a *model* for a KB $\mathcal{K}$ if it satisfies all assertions in $\mathcal{T}$ and $\mathcal{A}$.

[1]The extension of $(\exists R^-.A)^{\mathcal{I}}$ is defined in a similar way.

The use of an ontology language can be seen as an alternative to the use of standard modelling paradigms of ER [7] or UML[2] class diagrams. The advantage of an otology language over these formalisms lies on the fact that it has clear and unambiguous semantics which enable the use of automatic reasoning to support the designer. Indeed, $\mathcal{ELHI}$ ontology language is able to capture most conceptual modelling constructs, including relationships among classes, cardinality constraints in the participation of classes in relationships, and is-a relations among both, classes and relationships.

**Example 1.** Consider the ER diagram in Figure 1.1. The constraints expressed in the diagram can be represented in $\mathcal{ELHI}$ ontology using the following assertions:

| | | | |
|---|---|---|---|
| (1) | $\exists \mathsf{person\_role}.\mathsf{Role} \sqsubseteq \mathsf{Person}$ | (4) | $\mathsf{ActingRole} \sqsubseteq \mathsf{Role}$ |
| (2) | $\exists \mathsf{person\_role}^{-}.\mathsf{Person} \sqsubseteq \mathsf{Role}$ | (5) | $\mathsf{Actor} \sqsubseteq \mathsf{Person}$ |
| (3) | $\mathsf{Role} \sqsubseteq \exists \mathsf{person\_role}^{-}.\mathsf{Person}$ | | |

Assertions (1) and (2) correspond to the role typing (domain and range)[3] of relationship $\mathsf{person\_role}$, stating, respectively, that the first component of $\mathsf{person\_role}$ is of type $\mathsf{Person}$, while the second, i.e., its inverse, is of type $\mathsf{Role}$. Assertion (3) instead states mandatory participation for $\mathsf{Role}$ to relationship $\mathsf{person\_role}$ (4) and (5) express is-a relation among the respective classes.

## 2.2 Queries over $\mathcal{ELHI}$ Ontologies

To formulate queries over the ontology $\mathcal{K}$ we use conjunctive queries. A *conjunctive query* (CQ) $q$ over the KB $\mathcal{K}$ is an expression of the form $q(\boldsymbol{x}) \leftarrow body(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x}$ are the so-called *distinguished variables*, $\boldsymbol{y}$ are existentially quantified variables called *non-distinguished variables*, $q$ is a new predicate and $body(\boldsymbol{x}, \boldsymbol{y})$ is a conjunction of atoms of the form $A(x)$ or $R(x, y)$, where $A$ and $R$ are respectively an atomic concept and a role from $\mathcal{K}$, and $x$, $y$ are either constants in $\mathcal{K}$ or variables. $q(\boldsymbol{x})$ is called the *head* of the conjunctive query, and $body(\boldsymbol{x}, \boldsymbol{y})$ the *body*. Given a CQ $q$ and a KB $\mathcal{K}$, the *answer to $q$ over $\mathcal{K}$* is the set of tuples $q^{\mathcal{I}}$ of constants that substituted to $\boldsymbol{x}$ make the formula $\exists \boldsymbol{y}.body(\boldsymbol{x}, \boldsymbol{y})$ true in $\mathcal{I}$.

## 2.3 Wrapping Relational Sources

The purpose of data wrapping ontologies is to access data from relational sources. To this end, some of the concepts and roles are mapped to data stored in a database and described by a relational schema. We assume the reader is familiar with the basic notions of relational databases [1]. We consider a fixed denumerable alphabet $\Gamma_V$ of value constants and we consider databases over $\Gamma_V$. A *relational schema* $\mathcal{R}$ consists of *relation* symbols, and a *database instance* (or simply *database*) $\mathcal{D}$ over $\mathcal{R}$ is a set of relations with constants from $\Gamma_V$ as atomic values.

We adopt an ontology to relational data mapping schema in the spirit of the framework presented in [18] for linking data to ontologies. For this purpose, we introduce a new alphabet $\Lambda$ of function symbols in $\mathcal{ELHI}$, where each function symbol has an associated arity. Then, we define the set $\tau(\Gamma_V, \Lambda)$ of all *function terms* of the form $f(v_1, \ldots v_n)$ such that $f \in \Lambda$ with

[2]`http://www.uml.org`

[3]The fillers for $\mathsf{person\_role}$ role are not necessary for role typing and mandatory constraints expressed within the assertions (1) – (3).

arity $n > 0$, and $v_1, \dots v_n \in \Gamma_V$. We require that the set $\Gamma_O$ used to denote objects in $\mathcal{ELHI}$ coincides with $\tau(\Gamma_V, \Lambda)$. This implies that different function terms in $\tau(\Gamma_V, \Lambda)$ are interpreted as different objects in $\Delta$. Then, we define an *$\mathcal{ELHI}$ data wrapping ontology with mappings* as a triple $\mathcal{K}_m = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$, where $\mathcal{T}$ is an $\mathcal{ELHI}$ TBox, $\mathcal{D}$ is a database over the relational schema $\mathcal{R}$, and $\mathcal{M}$ is a set of *mapping assertions* of the form $\phi(\boldsymbol{f(x)}) \rightsquigarrow \psi(\boldsymbol{x})$, with $\psi$ an arbitrary (SQL) query over $\mathcal{D}$, $\phi$ an atom of the form $A(f(\boldsymbol{x}))$ or $R(f_1(\boldsymbol{x_1}), f_2(\boldsymbol{x_2})))$, $A$ and $R$, respectively, an atomic concept and role in $\mathcal{T}$, and $f(\boldsymbol{x})$ a variable term with $f$ in $\Lambda$ and $\boldsymbol{x}$ tuple of variables.

Given a source relational schema $\mathcal{R}$, the data wrapping ontology with mappings $\mathcal{K}_m = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ is derived form $\mathcal{R}$ in such a way that for each relation $r \in \mathcal{R}$, there are either atomic concepts $A_{db}$ and $A$ or roles $R_{db}$ and $R$[4], with $A_{db} \sqsubseteq A$ (respectively, $R_{db} \sqsubseteq R$), such that $A_{db}$ ($R_{db}$) has an associated mapping specifying how to retrieve the data about $A_{db}$ ($R_{db}$) from the data sources. By introducing two concepts for a relation, say $A_{db}$ and $A$, with a corresponding assertion $A_{db} \sqsubseteq A$, we "fix" the *db* subscripted terms as coming from the data sources, i.e., a syntactic convenience to be used in the emptiness testing algorithm; however, they both mean the same concept. Hence, in the rest of the paper, we will refer to such terms as *data source terms* and denote them by $\Sigma_{\mathcal{DB}}$. For details on the semantics of the mappings we refer to [18].

**Example 2.** For instance, the following mappings are associated to terms Person, Role and person_role from Figure 1.1[5].

$\mathsf{Person}(\texttt{pers}(id)) \rightsquigarrow$ `SELECT` *id* `FROM` *name*
$\mathsf{Role}(\texttt{role}(id)) \rightsquigarrow$ `SELECT` *id* `FROM` *role_type*
$\mathsf{person\_role}(\texttt{pers}(person\_id), \texttt{role}(role\_id)) \rightsquigarrow$
`SELECT` *person_id,role_id* `FROM` *cast_info*

## 2.4 Virtual ABox

We next introduce the notion of a virtual ABox, whose assertions are generated by "compiling" the mapping assertions starting from the data in $\mathcal{D}$.

**Definition 1.** Given a data wrapping ontology with mappings $\mathcal{K}_m = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ and a mapping assertion $M : \phi(\boldsymbol{f(x)}) \rightsquigarrow \psi(\boldsymbol{x})$ in $\mathcal{M}$, a *virtual ABox generated by $M$ from $\mathcal{D}$* is the set of assertions of the form $\mathcal{A}(M, \mathcal{D}) = \{\phi[\boldsymbol{f(x)}/\boldsymbol{f(v)}] \mid \boldsymbol{v} \in ans(\psi, \mathcal{D})\}$, where $\phi[\boldsymbol{f(x)}/\boldsymbol{f(v)}]$ is a ground atom, obtained from $\phi(\boldsymbol{f(x)})$ by substituting the $n$-tuple of variable terms $\boldsymbol{f(x)}$ with the $n$-tuple of constant terms $\boldsymbol{f(v)}$. Then, the *virtual ABox for $\mathcal{K}_m$* is the set of assertions $\mathcal{A}(\mathcal{M}, \mathcal{D}) = \{\mathcal{A}(M, \mathcal{D}) \mid M \in \mathcal{M}\}$.

Observe that by construction, $\mathcal{A}(\mathcal{M}, \mathcal{D})$ is over the constants $\tau(\Gamma_V, \Lambda)$ but represents all the data stored in the data sources $\mathcal{D}$[6]. Also, all concept and role names appearing in $\mathcal{A}(\mathcal{M}, \mathcal{D})$ are data source terms from $\Sigma_{\mathcal{DB}}$. Thus, we will consider $\mathcal{A}(\mathcal{M}, \mathcal{D})$ as an *incomplete database.*

It follows, from the semantics of the mappings [18] and the construction of the virtual ABox, that the models of $\mathcal{ELHI}$ data wrapping ontology with mappings $\mathcal{K}_m = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ and those

[4]Depending on integrity constraints imposed on relations in $\mathcal{R}$, there might be additional roles in the ontology $\mathcal{K}_m$; see [15].

[5]For the sake of simplicity, we don't explicitly display in Figure 1.1 the corresponding *db* terms. As mentioned, all entities and relationships in the middle layer are fixed to be database terms.

[6]projected on key attributes for the sake of simplicity

of $\mathcal{ELHI}$ ontology with virtual ABox $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}(\mathcal{M}, \mathcal{D}) \rangle$ coincide, i.e., we can express the semantics of $\mathcal{K}_m$ in terms of $\mathcal{ELHI}$ ontology with a virtual Abox. This immediately implies an algorithm to answer queries over an $\mathcal{ELHI}$ ontology with mappings: *(i)* compute $\mathcal{A}(\mathcal{M}, \mathcal{D})$, and *(ii)* apply query answering technique of [17] for $\mathcal{ELHI}$ ontology $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}(\mathcal{M}, \mathcal{D}) \rangle$. Given this, in the rest of the paper, we will only consider an $\mathcal{ELHI}$ ontology and a virtual ABox as a source (incomplete) database. Notice however that for the actual emptiness testing algorithm we will not explicitly build the virtual ABox; nevertheless, we will use this notion for our technical development described in the next section.

## 2.5 Emptiness of Ontology Terms

We now turn our attention to defining when a given predicate in an ontology is empty w.r.t. the information at the sources. From now on, we will consider a scenario where an $\mathcal{ELHI}$ ontology $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}(\mathcal{M}, \mathcal{D}) \rangle$ derived form the data sources, i.e., defined, roughly, over the signature $\Sigma_{\mathcal{DB}}$, has been enriched by adding to $\mathcal{T}$ new terms and/or assertions.

Given a term $\eta$ in $\mathcal{T}$ of an ontology $\mathcal{K}$, we call a *query for* $\eta$ a CQ $q(x) \leftarrow \eta(x)$ for $\eta$ an atomic concept and $q(x, y) \leftarrow \eta(x, y)$ for $\eta$ a role in $\mathcal{T}$. Our goal is to test whether $\eta$ is empty w.r.t. the data at the sources, i.e., w.r.t. $\Sigma_{\mathcal{DB}}$. Clearly, such a test should involve the query answering process; thus we now define the notion of query answering in the presence of an incomplete database.

**Definition 2.** Given an $\mathcal{ELHI}$ ontology $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}(\mathcal{M}, \mathcal{D}) \rangle$, and a query $q$ over $\mathcal{K}$, the *certain answers* to $q$ w.r.t. $\mathcal{K}$, denoted $cert(q, \mathcal{K})$, are the set of tuples $\boldsymbol{t}$ such that $\boldsymbol{t} \in q^{\mathcal{I}}$ for every interpretation $\mathcal{I}$ that is a model for $\mathcal{K}$.

Then, we say that a given term is empty, if the certain answers to its corresponding query are empty for *every* incomplete database $\mathcal{A}(\mathcal{M}, \mathcal{D})$.

**Definition 3.** Given an $\mathcal{ELHI}$ ontology $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}(\mathcal{M}, \mathcal{D}) \rangle$ and $\eta$ a term in $\mathcal{T}$ with query $q$ for $\eta$, we say that $\eta$ *is empty w.r.t.* $\Sigma_{\mathcal{DB}}$ iff $cert(q, \mathcal{K}) = \emptyset$ for every database $\mathcal{A}(\mathcal{M}, \mathcal{D})$.

This defines the problem studied in this paper: given a term $\eta \in \mathcal{T}$ with a CQ $q$ for $\eta$, test whether $cert(q, \mathcal{K}) = \emptyset$ for each $\mathcal{A}(\mathcal{M}, \mathcal{D})$. Note however that this does not imply that we will be necessarily computing $cert(q, \mathcal{K})$.

It is well known that the problem of computing answers in the presence of an incomplete database is often solved via *query rewriting* under constraints. Specifically, based on [17] we have that given a query $q$ over an $\mathcal{ELHI}$ ontology $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}(\mathcal{M}, \mathcal{D}) \rangle$, we can compute another query, a rewriting of $q$ denoted by $rew(q, \mathcal{T})$, such that the answers of $q$ over $\mathcal{K}$ and the answers of $rew(q, \mathcal{T})$ over $\mathcal{A}(\mathcal{M}, \mathcal{D})$ only coincide, i.e., $cert(q, \mathcal{K}) = rew(q, \mathcal{T})^{\mathcal{A}(\mathcal{M}, \mathcal{D})}$. Thus, we have the following:

**Theorem 1.** *Given an $\mathcal{ELHI}$ ontology $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}(\mathcal{M}, \mathcal{D}) \rangle$ and $\eta$ a term in $\mathcal{T}$ with query $q$ for $\eta$, $\eta$* is empty w.r.t. $\Sigma_{\mathcal{DB}}$ *iff $rew(q, \mathcal{T})^{\mathcal{A}(\mathcal{M}, \mathcal{D})} = \emptyset$ for every database $\mathcal{A}(\mathcal{M}, \mathcal{D})$.*

The above theorem shows that the problem of testing emptiness of a given term amounts to verifying whether the rewriting of its query returns empty answer for every possible database. We will see later that for this purpose we will not need to compute the actual evaluation, which makes our technique efficient in practice; however, we will employ the above relationship as described in the sequel.

# Chapter 3

# Emptiness Testing and Repair

## 3.1 Testing Emptiness

As follows from the previous section, to test emptiness of a given term we have to rewrite its corresponding query and check whether the obtained rewriting results in being empty. Recent work on rewriting conjunctive queries over $\mathcal{ELHI}$ ontologies [17] shows that for a CQ $q$ over $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}(\mathcal{M}, \mathcal{D}) \rangle$, $rew(q, \mathcal{T})$ is a Datalog program. Therefore, according to Theorem 1, our problem now comes down to testing emptiness of a query predicate $q$ in the rewritten Datalog program.

We first define some notions that will be needed throughout this section.

### 3.1.1 Preliminary Notions

A *Datalog program* $\Pi$ over an $\mathcal{ELHI}$ ontology $\mathcal{K}$ consists of *(i)* a set of rules of the form $head(\boldsymbol{x}) \leftarrow body(\boldsymbol{x}, \boldsymbol{y})$, where $body(\boldsymbol{x}, \boldsymbol{y})$ is a conjunction of atoms involving concept and role names in $\mathcal{K}$; *(ii)* a *special rule* that is a CQ with a *query predicate* $q$ in the head.

The *extensional database (EDB) predicates* of a Datalog program $\Pi$ are those that do not appear in the head of any rule in $\Pi$, all other predicates are called *intentional database (IDB) predicates*. Given a (incomplete) database $\mathcal{A}(\mathcal{M}, \mathcal{D})$, the evaluation $\Pi(\mathcal{A}(\mathcal{M}, \mathcal{D}))$ of $\Pi$ over an EDB $\mathcal{A}(\mathcal{M}, \mathcal{D})$, is the evaluation of the special rule, denoted by $q_\Pi(\mathcal{A}(\mathcal{M}, \mathcal{D}))$, taken as a CQ, over the minimum Herbrand model of $\Pi \cup \mathcal{A}(\mathcal{M}, \mathcal{D})$ [1].

Given a Datalog program $\Pi$ and an IDB predicate $q$ in $\Pi$, the associated *AND-OR tree for $q$ in* $\Pi$ is a set of labelled nodes such that *(i)* the root of the tree is a (and-)node labelled by $q$; *(ii)* for every and-node labelled by $g_i$, and for every rule $r_i$ of $\Pi$ having $g_i$ in the head, there exists an or-node, child of $g_i$, labelled by $r_i$; *(iii)* for every or-node labelled with a rule $r_i$ in $\Pi$, and for every atom name $g_{i_j}$ in the body of $r_i$, there exists an and-node labelled with $g_{i_j}$.

An *or-branch* of an AND-OR tree is a set of and-nodes $\mathcal{G}$ that are children of a unique combination of or-nodes of the tree, i.e., when several sibling or-nodes are present, only children of one of the or-nodes are contained in $\mathcal{G}$.

### 3.1.2 Datalog Predicate Emptiness: Known Results

In [20], Vardi discussed the problem of deciding emptiness of IDB predicates in Datalog programs and showed it to be decidable in polynomial time by building the so-called skeletons of expansion trees, $skel(P, \Pi)$, for a given IDB predicate $P$, and constructing a tree automaton whose set of accepted trees is $skel(P, \Pi)$ (the nonemptiness problem for tree automata is decidable in polynomial time).

The key idea underlying this result is the observation that a Datalog program can be viewed as an infinite union of CQs. That is, for each IDB predicate $P$ in $\Pi$ there is an infinite sequence $C_0, C_1, \ldots$ of CQs such that for every EDB $\mathcal{A}(\mathcal{M}, \mathcal{D})$, $P_\Pi(\mathcal{A}(\mathcal{M}, \mathcal{D})) = \bigcup_{i=0}^{\infty} C_i(\mathcal{A}(\mathcal{M}, \mathcal{D}))$. The $C_i$s are called the *expansions* of $P$ and can be described in terms of *expansion trees*. Roughly, the root of an expansion tree for an IDB predicate $P$ is labelled with $P(\boldsymbol{x})$ atom and expanded to its child nodes labelled with the body atoms of the rule for $P$. Each child node labelled with IDB atom $P_i(\boldsymbol{x_i})$ is in turn expanded with nodes labelled with the body atoms of the rule whose head atom unifies with $P_i(\boldsymbol{x_i})$. Each child node labelled instead with EDB atom is a leaf. The CQ corresponding to such a tree is the conjunction of all EDB atoms for all leaves in the tree. Then it immediately follows that, given a set of all expansion trees for an IDB $P$ in $\Pi$, $trees(P, \Pi)$, we have that $P$ is empty in $\Pi$ iff $trees(P, \Pi)$ is empty, i.e. all the corresponding queries for $trees(P, \Pi)$ are empty.

The problem with expansion trees is clearly the fact that the set of labels on the nodes of the expansion trees is potentially infinite, due to recursive rules in a Datalog program. However, [20] shows that we can get rid of variables when building expansion trees, obtaining skeletons of expansion trees $skel(P, \Pi)$ that are labelled with only predicate symbols, and still having the property of $P$ being empty in $\Pi$ iff $skel(P, \Pi)$ is empty.

### 3.1.3 Emptiness Testing Algorithm

We build our approach on the results of [20], and in particular on the possibility of building finitely labelled trees for IDB predicates. Note that while [20] presents the problem as decision problem on emptiness of tree automata ([20] is specifically tailored for the exposition of the use of tree automata), we prefer to present direct algorithms (that do not involve tree automata) because working with skeleton trees, as we will see, is conceptually much simpler.

Given a term $\eta$ with a CQ $q$ for $\eta$ in an $\mathcal{ELHI}$ ontology $\mathcal{K}$, we devise our emptiness testing algorithm in four steps:

1. rewrite $q$ using procedure of [17], obtaining a Datalog program $\Pi$,
2. add to $\Pi$ auxiliary rules for making IDB predicates explicit,
3. for the resulting Datalog program with a query predicate $q$, build an AND-OR tree for $q$, and
4. visit the obtained tree and mark its nodes as empty/nonempty corresponding to empty/nonempty predicates, which, in turn, correspond to empty/nonempty concepts and roles in $\mathcal{K}$.

In the following we will elaborate on steps *(ii)-(iv)*, for details on the rewriting algorithm we refer to [17].

## Adding auxiliary rules to $\Pi$

Consider a Datalog program $\Pi$ resulting from rewriting a CQ for a given term over an ontology $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}(\mathcal{M}, \mathcal{D})\rangle$. Then for each term $\eta$ that is not among database terms in $\Sigma_{\mathcal{DB}}$ and does not appear in the head of any rules of $\Pi$, add to $\Pi$ an auxiliary rule $\eta(x) \leftarrow \eta(x)$ (resp. $\eta(x,y) \leftarrow \eta(x,y)$) for $\eta$ corresponding to an atomic concept (resp. role) in $\mathcal{T}$ of $\mathcal{K}$. We denote the resulting Datalog program by $\Pi^*$.

The intuition here is that, since, by construction of the data wrapping ontology in Section 2.3, all terms subscripted with *db* appear only on the left-hand sides of inclusion assertions, then, by virtue of the rewriting algorithm [17], the corresponding predicates in $\Pi$ won't be saturated and thus are guaranteed to occur only in the bodies of the rules of $\Pi$ (i.e., as EDB predicates). However, it is not the case that the rest of the terms of $\mathcal{T}$ (i.e., non database terms) occur only in the heads of the rules of the rewritten program $\Pi$. Therefore, using auxiliary rules above, we explicitly make all non database terms as IDB predicates. Note that an auxiliary rule $\eta(x) \leftarrow \eta(x)$ is equivalent to a tautology $\eta(x) \vee \neg\eta(x)$. Thus, from a logical point of view, we do not change the semantics of the program $\Pi$. That is, an IDB predicate $q$ is empty in $\Pi$ iff $q$ is empty in $\Pi^*$, with $\mathcal{A}(\mathcal{M}, \mathcal{D})$ considered as an EDB, i.e., $q_{\Pi}(\mathcal{A}(\mathcal{M}, \mathcal{D})) = \emptyset$ iff $q_{\Pi^*}(\mathcal{A}(\mathcal{M}, \mathcal{D})) = \emptyset$

## Building skeleton tree for $q$ in $\Pi^*$

The *skeleton tree* for a query predicate $q$ in $\Pi^*$, $skel(q, \Pi^*)$, is an AND-OR tree for $q$ in $\Pi^*$ (we assume all rules are named beforehand), with a condition that an and-node is not expanded (i.e., is a leaf), if either

- it is labelled with an EDB predicate,
- it has an isomorphic and-node (i.e., a node labelled with the same predicate symbol) that has already been expanded, or
- it is marked as empty/nonempty i.e., has already been processed before (in another skeleton tree).

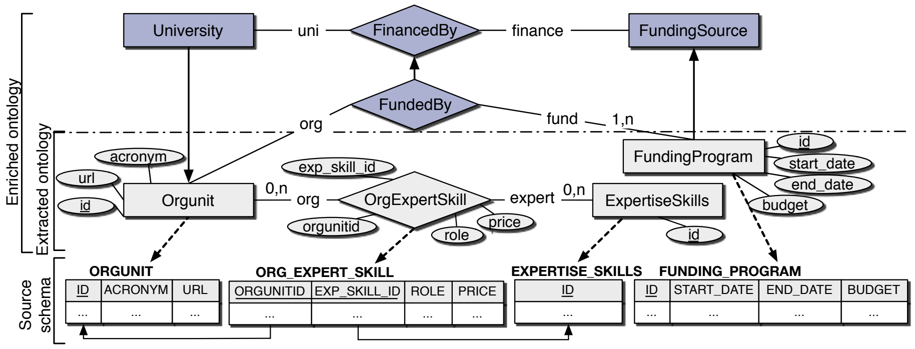


Figure 3.1: Example of data wrapping ontology

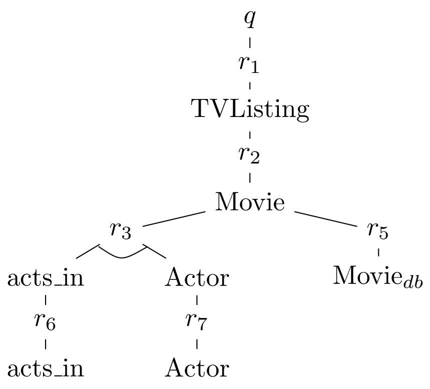


Figure 3.2: Skeleton tree corresponding to the Datalog program of Example 3

**Example 3.** Consider the wrapping ontology in Figure 3.1 and consider the (part of a) Datalog program with rules $r_1$ to $r_7$ below obtained by rewriting a CQ $q(x) \leftarrow \mathsf{TVListing}(x)$, i.e., we want to test emptiness of TVListing. We additionally have auxiliary rules $r_6$ and $r_7$ to make acts_in and Actor IDB predicates.

$r_1 : q(x) \leftarrow \mathsf{TVListing}(x)$
$r_2 : \mathsf{TVListing}(x) \leftarrow \mathsf{Movie}(x)$
$r_3 : \mathsf{Movie}(x) \leftarrow \mathsf{acts\_in}(y, x), \mathsf{Actor}(y)$
$r_4 : \mathsf{Person}(x) \leftarrow \mathsf{Actor}(x)$
$r_5 : \mathsf{Movie}(x) \leftarrow \mathsf{Movie}_{db}(x)$
$r_6 : \mathsf{acts\_in}(x, y) \leftarrow \mathsf{acts\_in}(x, y)$
$r_7 : \mathsf{Actor}(x) \leftarrow \mathsf{Actor}(x)$.

The skeleton tree for this Datalog program is shown in Figure 3.2. Note that the children of $r_6$ and $r_7$, the and-nodes acts_in and Actor are not expanded anymore, since they have isomorphic nodes that have already been expanded.

Note that the size of the skeleton tree, as well as the time needed to build it, is linear in the number of the rules in $\Pi^*$, since, by construction of the tree, each rule in $\Pi^*$ is expanded *only once*. Moreover, it is immediate to see that an AND-OR skeleton tree obtained in this way represents all the skeletons of expansion trees as defined in [20]. So, a predicate $q$ is empty, iff all or-branches of $skel(q, \Pi^*)$ are empty. We next define emptiness of nodes in $skel(q, \Pi^*)$.

### Visiting skeleton tree

Once the tree is built, our algorithm examines it bottom-up (depth-first) and marks the respective nodes as empty or nonempty[1]. Specifically, starting from an and-leaf labelled with $g_{i_j}$, if $g_{i_j}$ is an EDB predicate in $\Pi^*$, then it is marked as nonempty. The algorithm then visits next sibling of $g_{i_j}$ and checks for its emptiness/nonemptiness, so that a parent of $g_{i_j}$, an or-node labelled by a rule $r_i$, is marked as nonempty iff *all* its children are marked as nonempty. The algorithm then proceeds to an and-node $g_i$, parent of $r_i$, marking it as nonempty iff *at least one* of its children or-nodes are nonempty.

**Example 4.** Consider the skeleton tree in Figure 3.2 for the query predicate $q$ of Datalog program of Example 3. We start with acts_in leaf and mark it as empty (it is non EDB

[1]A node can also be temporarily marked as *unknown* but we skip the details here.

predicate). This makes also its parent $r_6$ and, in turn, $\mathsf{acts\_in}$ empty. To decide for $r_3$, we have to check the or-branch starting with $\mathsf{Actor}$, which results in being empty. Therefore, $r_3$ is empty. Again, to decide for $\mathsf{Movie}$, we look at the or-branch on the right-hand side. $\mathsf{Movie}_{db}$ is an EDB predicate, so it is nonempty. Consequently, we mark $r_5$ and $\mathsf{Movie}$ as nonempty, which determines non-emptiness for $r_2$ and then $\mathsf{TVListing}$, $r_1$ and finally $q$. Indeed, we can construct a CQ $q(x) \leftarrow \mathsf{Movie}_{db}(x)$ that guarantees non-emptiness when evaluated over the actual data.

It is important to note that emptiness of a node is "global", meaning that if a node is empty, it will be empty in every skeleton tree it appears in (and the same for non-emptiness). This is due to the rewriting algorithm [17] which "compiles" in the Datalog program all the knowledge about a given term. For this reason, as we have already mentioned, each predicate in a Datalog program is expanded only once.

Finally, notice that the technique proposed in this section is applicable to ontology languages in the full spectrum of DLs from $\mathcal{ELHI}$ to *DL-Lite*$_{core}$ [5]. This again because of the rewriting technique [17] of being able to deal with this range of languages. The rewriting of a CQ over a *DL-Lite* KB is a union of CQs [17], however, a Datalog program can be always viewed as a union of CQs or a single CQ.

## 3.2 Repairing Empty Terms

So far, we have devised a procedure for verifying whether a given term in a data wrapping ontology is empty w.r.t. the database terms at the sources. We next present a method for supporting the repair of empty concepts and roles, consisting of a set of *guidelines* for ontology engineers.

### 3.2.1 Generating Repair Guidelines

To suggest a repair for an empty term, we naturally resort to the Datalog program $\Pi^*$ and the skeleton tree generated from $\Pi^*$ by our emptiness testing algorithm. Indeed, the skeleton tree for a term $\eta$, by virtue of its construction, contains as nodes all and only relevant terms for $\eta$: those that contribute or could contribute to its non-emptiness. So an intuitive way to fix an empty term is to *focus* on the relevant nodes (in one of the or-branches) of its corresponding skeleton tree and to possibly *expand* those nodes by rendering them nonempty. The possible expansion should obviously be in correspondence with an addition or refinement of a term or/and assertion in the actual ontology. We elaborate on this idea in the rest of this section.

Given a skeleton tree constructed by the algorithm with an and-node $g$ in the tree, let $\mathcal{G}^* = [G_1, \ldots, G_n]$ denote the sequence of sets of its and-nodes, such that, intuitively, each $G_i$ contains, in a bottom-up fashion, distinct groupings of and-nodes in *one* of the or-branches of the tree that should be marked as nonempty in order for $g$ to be marked as nonempty. Moreover, $\mathcal{G}^*$ is such that, in order for and-nodes in $G_i$ to be marked as nonempty in the tree, all the $G_k$s, $k = \{1, \ldots, i-1\}$ have to be marked as nonempty.

**Example 5.** Suppose the rule $r_5$ was not present in the tree of Figure 3.2. Then, for an and-node $\mathsf{TVListing}$, there would be only one or-branch in the tree, $\mathcal{G}^* = [\{\mathsf{acts\_in}, \mathsf{Actor}\}, \{\mathsf{Movie}\}]$. The intuition here is that both, $\mathsf{acts\_in}$ and $\mathsf{Actor}$, and $\mathsf{Movie}$ have to be repaired in order for

$\mathsf{TVListing}$ to become nonempty. And similarly, to repair $\mathsf{Movie}$, both $\mathsf{acts\_in}$ and $\mathsf{Actor}$ must be rendered to be no longer nonempty.

Thus, for each and-node $g_{i_j}$ in every $G_i$, of every or-branch of the skeleton tree, our strategy is to consider $g_{i_j}$ as a leaf in the tree, and to examine its possible expansions, whereas to expand the leaf we mainly need a new rule with its corresponding atom in the head. Given such a rule, we can track down the needed terms and assertions in the ontology and provide those repairs as guidelines to the user. For this purpose, we distinguish two cases: *(i)* $g_{i_j}$ corresponds to an atomic concept $A$, and *(ii)* $g_{i_j}$ corresponds to a role $R$[2].

For case *(i)*, our repair service provides two guidelines. First, it suggests to add an inclusion assertion with $A$ on the right-hand side. This, from the modelling point of view, results in either defining participation constraints for $A$ to a relationship $R$, if $R$ appears in any of $G_k$s, $k = \{1, \ldots, i-1\}$, of $\mathcal{G}^*$, or asserting $A$ as a superclass of some class $B$, verifying beforehand that $B$ is nonempty. Second, if $B(x) \leftarrow A(x)$ is present in the program $\Pi^*$ and $B$ is nonempty, the user is warned with misplaced is-a relationship, i.e., maybe $B \sqsubseteq A$ should be added instead of $A \sqsubseteq B$.

For case *(ii)* we have again two possible guidelines. The first one hints to add an inclusion assertion between roles with $R$ on the right-hand side. The second, if a concept $A$ appears in any of $G_k$s as above, and $A$ is nonempty, the service suggests to add an assertion $A \sqsubseteq \exists R$, i.e., mandatory participation for $A$ in the relationship $R$.

**Example 6.** As can be seen, $\mathsf{Actor}$ and $\mathsf{acts\_in}$ terms are empty: they are not explicitly mapped to the database and cannot be rewritten to non empty terms. For $\mathsf{Actor}$, our repair service suggests to either assert it as a superclass to some (possibly still to be added) nonempty class, or to replace $\mathsf{Actor} \sqsubseteq \mathsf{Person}$ with $\mathsf{Person} \sqsubseteq \mathsf{Actor}$ which, evidently, in this case is not appropriate. To repair $\mathsf{acts\_in}$, the user will be suggested to either assert mandatory participation for $\mathsf{Movie}$ to relationship $\mathsf{acts\_in}$: $\mathsf{Movie} \sqsubseteq \exists \mathsf{acts\_in}^-$, or to make $\mathsf{acts\_in}$ more general than some (possibly still to be added) nonempty relationship.

Finally, if none of the above mentioned repairs are possible, we suggest to explicitly map to the sources either the actual empty term or its relevant terms corresponding to each $g_{i_j}$ in the skeleton tree.

[2]In the following, we will leave implicit the correspondence of expansion of an and-node with a rule in the Datalog program.

# Chapter 4

# Implementation and Evaluation

## 4.1 Tool for Emptiness Testing and Repair

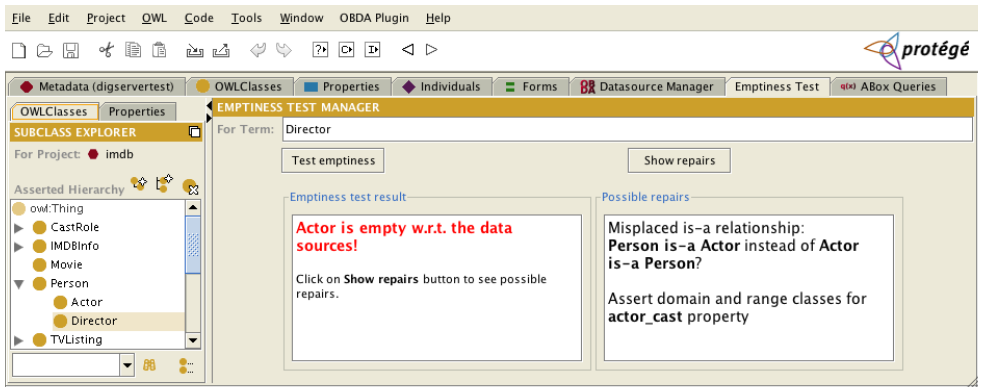
We have implemented our approach as a (preliminary) plug-in for Protégé[1] that enhances OBDA plug-in[2]. The OBDA plug-in provides facilities to design Ontology Based Data Access (OBDA) system components (i.e., data sources and mappings). It supports the definition of relational data sources and GAV-like mappings to link concepts and roles of the $DL\text{-}Lite_{\mathcal{A}}$ ontology [18] to data in the defined sources. It also allows for conjunctive query answering (using SPARQL syntax), a service commonly offered by OBDA centric reasoners.

The key goal of our plug-in is to provide a support for verifying emptiness of a selected term in an ontology w.r.t. the data at the defined sources, and for repairing empty terms by allowing the user to explore different repair solutions. Figure 4.1 shows the screenshot of our Protégé plug-in when using the wrapping ontology to query the data (using OBDA plug-in's *ABox Queries* tab) at the underlying database (specified together with mappings in *Datasource Manager* tab). As can be seen, for a selected class or property at the left segment of the plug-in's window, the user can verify, by clicking on *Test emptiness* button, whether that term will return any answer when queried against the source database (without computing the actual evaluation!). When a term results in being empty, the user can ask for guidelines to repair that term by clicking on *Show repairs* button. The repairs are then devised by the tool and shown in a text pane in natural language using standard ontology modelling terminology. The issue of presenting repairs and assisting a user in incorporating them into an ontology is the subject of future work.

## 4.2 Usability Study

In order to determine the practical use and efficiency of the emptiness testing and repair features implemented in Protégé plug-in, we conducted a small usability study. In this section we describe its procedure and results obtained.

[1]`http://protege.stanford.edu`
[2]`http://obda.inf.unibz.it/protege-plugin`



Actor is empty w.r.t. the data sources!

Click on **Show repairs** button to see possible repairs.

Misplaced is-a relationship:
**Person is-a Actor** instead of **Actor is-a Person**?

Assert domain and range classes for **actor_cast** property

Figure 4.1: Emptiness testing and repair in Protégé

### 4.2.1 Procedure

#### Participants

The experiment involved 10 subjects, all with good and homogeneous understanding of description logics reasoning, ontology-based data access, and experience using Protégé. Most subjects were graduate students (8 graduates and 2 undergraduate) that have attended courses on the mentioned topics and have used Protégé editor and the OBDA plug-in for their practical projects.

#### Database and ontology used

The domain used for the study was that of TV programmes. In particular, for the sources, we used a real database schema, IMDB movie database, retrieved using IMDbPY[3] and containing 21 relational tables. The original database schema was designed in a purely object-oriented fashion (e.g. table *cast_info* recording for each person in which movie he/she casts with which character and role (actor, director, etc.) was identified by an automatically generated ID and not the combination of the aforementioned attributes). Therefore, we annotated the database with keys.

The TVListings ontology was then automatically derived [15] from the annotated IMDB database together with mappings, and extended with terms and assertions to refine the movies domain (e.g., by adding Actor, Genre, etc.) and to (partly) describe TV programmes. The obtained ontology contained 24 classes and 14 properties; and can be found, in $DL\text{-}Lite_{\mathcal{A}}$ notation, in Appendix A. Note that while newly added concepts and roles specific to IMDB movies domain, as Actor and Genre, can be populated by specifying arbitrary SQL queries over IMDB database by asserting them in the mappings, this is not the case for "broader" terms, as TVListingKind, since these terms are not restricted to have instances stored in IMDB database.

[3]http://imdbpy.sourceforge.net/

### Tasks

The subjects were randomly divided into two groups, each of 5 subjects:

**group 1** received no support for testing emptiness of ontology terms and repairing them;

**group 2** could use the Protégé plug-in for testing emptiness of terms and ask for guidelines to repair empty terms.

Then, each subject was given four simple queries over $\mathsf{TVListings}$ ontology returning empty answers (we use here CQ notation, instead of SPARQL as actually used in OBDA plug-in):

**query 1:** $q(x) \leftarrow \mathsf{Actor}(x), \mathsf{actor\_cast}(x, y), \mathsf{CastCharacter}(y)$;

**query 2:** $q(x) \leftarrow \mathsf{hasGenre}(x, y)$;

**query 3:** $q(x, y) \leftarrow \mathsf{tvlisting\_info}(x, y)$;

**query 4:** $q(y) \leftarrow \mathsf{TVListing}(x), \mathsf{hasTVListingKind}(x, y), \mathsf{TVListingKind}(y)$.

Terms $\mathsf{Actor}$, $\mathsf{hasGenre}$, $\mathsf{tvlisting\_info}$, $\mathsf{hasTVListingKind}$ and $\mathsf{TVListingKind}$ are empty but are all repairable.

Given the wrapping ontology and the queries, the subjects were asked to add to the ontology new assertions so that the given queries were no longer empty. This involved identifying atoms responsible for query emptiness and repairing the corresponding terms. Note that while most subjects were obviously familiar with IMDB domain, none of them had seen before neither the ontology, nor the queries.

### Data collected

The following input was elicited during after the process of "fixing" empty queries:

- during the process, the subject was asked to write down a brief explanation for emptiness of the query, based on his/her understanding of the problem;
- the time needed to complete the tasks, as well as the number of changes made to the ontology during entire process was automatically recorded;
- after completing their tasks, the subjects in group 2 were asked to fill a questionnaire concerning their experience using the tool and give further comments (if any) regarding its usage;

### Goal

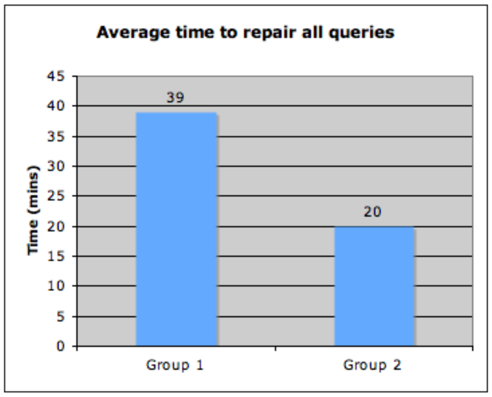
The goal of this study was twofold:

- compare the time taken to complete the tasks between the two groups;
- to evaluate overall user experience using the plug-in.

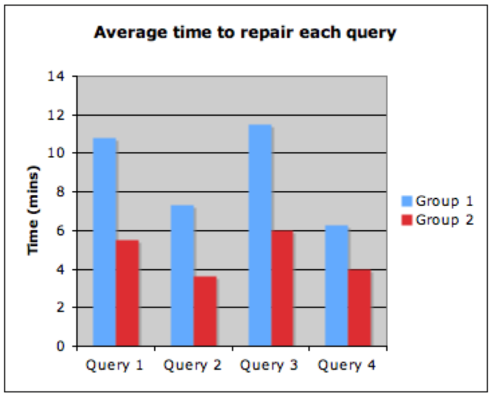
### 4.2.2 Results

#### Task times and changes made

Overall, as shown in Figure 4.2(a), the average time taken for group 1 was 39 minutes, and 20 minutes for group 2. The time to repair each of the given queries is shown in Figure 4.2(b).



(a) Average time to repair given queries



(b) Average time to repair each query

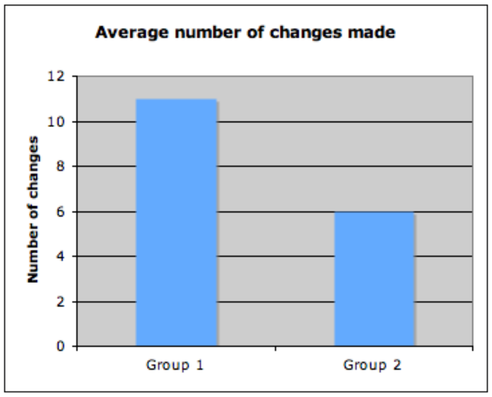
Figure 4.2: Times to repair given queries

The average number of changes made to the ontology in order to repair given queries, which we consider to be as key sub-task, for group 1 was 11, and 6 for group 2. This is also shown in Figure 4.3(a). Figure 4.3(b) instead displays the number of changes made to repair each of the given queries. The total number of changes needed for all queries was 5. This means that, in average, each subject in group 1 made 5 *erroneous changes* to repair the given queries, while in group 2 – 1 erroneous change.

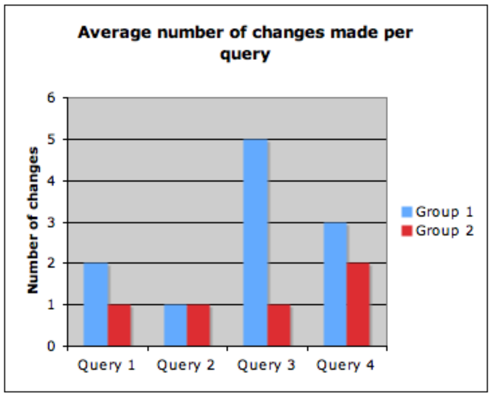
#### Post-study self-reported data

As mentioned, we have also collected user reactions to the tool. The questionnaire used for this purpose was composed of 10 short statements, each accompanied by a 5-point scale of "strongly disagree" (1 point) to "strongly agree" (5 points). Thus, given 5 subjects in group 2, each statement scores to maximum of 25 points. Table 4.2.2 provides for each statement its overall score.

The response of the users, also reflected in Table 4.2.2, points to the area of improvement w.r.t. the usability of the plug-in. In particular, the weakness of statement 2 suggests, besides manifesting that a term is empty, to add some kind of explanation for their emptiness. Similarly, the form of representation of repairs needs to be improved. Indeed, 2 subjects commented that the language used assumes good modelling understanding, which may not be the case. Additionally, when showing possible repairs for a given term, and in particular



(a) Average ontology changes made to repair given queries



(b) Average ontology changes made to repair each query

Figure 4.3: Changes made to repair given queries

| **Statement** | **Score** |
|---|---|
| 1. It was simple to use this tool. | 24 |
| 2. I could effectively identify the reason for query emptiness using this tool. | 19 |
| 3. I believe I could identify empty terms with this tool faster than without it. | 25 |
| 4. I could effectively "fix" empty queries using this tool. | 21 |
| 5. I believe I could repair empty terms with this tool faster than without it. | 25 |
| 6. I found repair guidelines to be adequate. | 21 |
| 7. I often incorporated repair guidelines to "fix" empty terms. | 24 |
| 8. The information given by the tool was easy to understand. | 20 |
| 9. This tool has all the functionalities I expect it to have. | 21 |
| 10. Overall, I am satisfied with this tool. | 25 |

Table 4.1: Overall rating of the plug-in

when suggesting to assert an empty term as a super-entity (or -relationship), 1 user suggested to add the list of likely terms for that.

Overall, the feedback of the subjects in the study was encouraging. Having a substantial background in the area of DL reasoning and ontology-based data access, they were still surprised how much easier and faster they could complete the given tasks.

# Chapter 5

# Related Work

To our knowledge, no research has been devoted for supporting the problem addressed in this paper. There are several ontology engineering methodologies in the literature (see [8] for a survey); but they are mostly focused on the design of what we call domain ontologies rather than accessing (relational) data sources. Note that the techniques proposed in this paper can be used with most of the well established methodologies.

Recently there have been several contributions to the problem of debugging and repairing unsatisfiable concepts in DL ontologies (see e.g. [11, 4]); however, the problem that we address is different so these techniques cannot be applied.

As described in Section 3.1, our core algorithm is strictly related to the problem of emptiness of intensional predicates in Datalog programs (see e.g. [13]). However, those techniques cannot be applied directly because of the fact that we adopt the DL $\mathcal{ELHI}$ instead of Datalog. The former is better suited for characterising the kind of axioms required for capturing common ER/UML constructs, and part of the upcoming OWL2 W3C recommendation.[1]

It is worth to mention the work in [16] as related, where M. Marx shows that the packed fragment of FOL admits view-based rewritings. This means that given a FOL fragment and a database associated to some of its predicates, an arbitrary FOL query involving only implicitly defined predicates can be executed directly as an SQL query over the database extended with the pre-computed materialised views that encode the explicit definitions. The restriction to the views is however in some cases too strong, as for instance TVListing in Figure 3.1 is not *definable* in terms of views but can be *rewritten* to a data source term.

[1]`http://www.w3.org/TR/2009/CR-owl2-profiles-20090611/`

# Chapter 6

# Conclusions and Future Work

This paper presents a technique for supporting ontology engineers in the development of ontologies for accessing relational data sources. We introduced the problem of deciding the emptiness of a given query w.r.t. a DL theory where data can be accessed only through a subset of the concepts and roles (analogously to the EDB/IDB predicates distinction in Datalog programs). Moreover, we have shown how the algorithm to decide the above problem can be exploited in order to support the engineer in “repairing” the ontology.

We enhanced the OBDA Protégé plug-in in order to support our technique and we evaluated its effectiveness and usability with an experiment involving external users.

The algorithm presented can be also applied in other scenarios, for example, for *optimising* the rewriting. Indeed, rules with empty predicates in the rewriting will not contribute to an answer, and thus can be eliminated. For instance, rule $r_3$ in Example 3 can be removed from the program: when evaluated against the actual data, it won’t return any answer.

The next direction for the work reported in this paper is the possibility of testing predicate emptiness in programs with function symbols. If this was feasible, the rewriting step would not be needed anymore.

# Appendix A

# TVListings Ontology

$\mathsf{CastingInfo} \sqsubseteq \exists\mathsf{listsCastCharacter}$
$\mathsf{CastingInfo} \sqsubseteq \exists\mathsf{listsCastRole}$
$\mathsf{CastingInfo} \sqsubseteq \exists\mathsf{listsMovie}$
$\mathsf{CastingInfo} \sqsubseteq \exists\mathsf{listsPerson}$
$\mathsf{ActingRole} \sqsubseteq \mathsf{CastRole}$
$\mathsf{DirectingRole} \sqsubseteq \mathsf{CastRole}$
$\mathsf{WritingRole} \sqsubseteq \mathsf{CastRole}$
$\mathsf{DistributionCompany} \sqsubseteq \mathsf{Company}$
$\mathsf{ProductionCompany} \sqsubseteq \mathsf{Company}$
$\mathsf{Genre} \sqsubseteq \mathsf{IMDBInfo}$
$\mathsf{Rating} \sqsubseteq \mathsf{IMDBInfo}$
$\mathsf{MovieCompaniesInfo} \sqsubseteq \exists\mathsf{involvesCompany}$
$\mathsf{MovieCompaniesInfo} \sqsubseteq \exists\mathsf{involvesMovie}$
$\mathsf{MovieCompaniesInfo} \sqsubseteq \exists\mathsf{withCompanyType}$
$\mathsf{Actor} \sqsubseteq \mathsf{Person}$
$\mathsf{Actor} \sqsubseteq \exists\mathsf{actor\_cast.CastCharacter}$
$\mathsf{Director} \sqsubseteq \mathsf{Person}$
$\mathsf{Writer} \sqsubseteq \mathsf{Person}$
$\mathsf{Movie} \sqsubseteq \mathsf{TVListing}$
$\exists\mathsf{hasMovieKind} \sqsubseteq \mathsf{Movie}$
$\exists\mathsf{hasMovieKind}^{-} \sqsubseteq \mathsf{MovieKind}$
$\exists\mathsf{tvlisting\_info} \sqsubseteq \mathsf{TVListing}$
$\mathsf{TVListing} \sqsubseteq \exists\mathsf{tvlisting\_info}$
$\exists\mathsf{tvlisting\_info}^{-} \sqsubseteq \mathsf{TVListingInfo}$
$\exists\mathsf{hasGenre} \sqsubseteq \mathsf{Movie}$
$\exists\mathsf{hasGenre}^{-} \sqsubseteq \mathsf{Genre}$
$\exists\mathsf{movie\_info} \sqsubseteq \mathsf{Movie}$
$\exists\mathsf{tvlisting\_info}^{-} \sqsubseteq \mathsf{MovieInfo}$
$\exists\mathsf{person\_info} \sqsubseteq \mathsf{Person}$
$\exists\mathsf{person\_info}^{-} \sqsubseteq \mathsf{IMDBInfo}$
$\exists\mathsf{listsMovie} \sqsubseteq \mathsf{CastingInfo}$
$\exists\mathsf{listsMovie}^{-} \sqsubseteq \mathsf{Movie}$
$\exists\mathsf{listsCastCharacter} \sqsubseteq \mathsf{CastingInfo}$
$\exists\mathsf{listsCastCharacter}^{-} \sqsubseteq \mathsf{CastCharacter}$
$\exists\mathsf{listsCastRole} \sqsubseteq \mathsf{CastingInfo}$
$\exists\mathsf{listsCastRole}^{-} \sqsubseteq \mathsf{CastRole}$
$\exists\mathsf{listsPerson} \sqsubseteq \mathsf{CastingInfo}$
$\exists\mathsf{listsPerson}^{-} \sqsubseteq \mathsf{Person}$
$\exists\mathsf{involvesCompany} \sqsubseteq \mathsf{MovieCompaniesInfo}$
$\exists\mathsf{involvesCompany}^{-} \sqsubseteq \mathsf{Company}$
$\exists\mathsf{involvesMovie} \sqsubseteq \mathsf{MovieCompaniesInfo}$
$\exists\mathsf{involvesMovie}^{-} \sqsubseteq \mathsf{Movie}$
$\exists\mathsf{withCompanyType} \sqsubseteq \mathsf{MovieCompaniesInfo}$
$\exists\mathsf{withCompanyType}^{-} \sqsubseteq \mathsf{CompanyType}$

Additionally, there are TVListingKind concept and hasTVListingKind role in the ontology for which no constraints are defined.

# Bibliography

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] F. Baader. Terminological cycles in a description logic with existential restrictions. In *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 325–330, 2003.

[3] F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 364–369, 2005.

[4] Franz Baader and Rafael Penaloza. Axiom pinpointing in general tableaux. *Journal of Logic and Computation*, 2008.

[5] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The dl-lite family. *J. of Automated Reasoning*, 39(3):385–429, 2007.

[6] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Data integration in data warehousing. *Int. J. of Cooperative Information Systems*, 10(3):237–271, 2001.

[7] P. Chen. The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976.

[8] Óscar Corcho, Mariano Fernández-López, and Asunción Gómez-Pérez. Methodologies, tools and languages for building ontologies: Where is their meeting point? *Data and Knowledge Engineering*, 46(1):41–64, 2003.

[9] J. Euzenat and P. Shvaiko. *Ontology Matching.* Springer-Verlag, 2007.

[10] J. Heflin and J. Hendler. A portrait of the Semantic Web in action. *IEEE Intelligent Systems*, 16(2):54–59, 2001.

[11] A. Kalyanpur, B. Parsia, E. Sirin, and B. Cuenca-Grau. Repairing unsatisfiable concepts in OWL ontologies. In *The Semantic Web: Research and Applications, 3rd European Semantic Web Conference (ESWC 2006)*, pages 170–184. Springer, 2006.

[12] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of the 21st ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems (PODS 2002)*, pages 233–346. ACM, 2002.

[13] A. Y. Levy. *Irrelevance Reasoning in Knowledge Based Systems.* PhD thesis, Stanford University, 1993.

[14] L. Lubyte and S. Tessaris. Supporting the design of ontologies for data access. In *Workshop Notes of the Int. Workshop on Description Logics (DL 2008)*. CEUR Electronic Workshop Proceedings, 2008.

[15] L. Lubyte and S. Tessaris. Automatic extraction of ontologies wrapping relational data sources. In *Proc. of the 20th Int. Conf. on Database and Expert Systems Applications*, 2009. To appear.

[16] M. Marx. Queries determined by views: Pack your views. In *Proc. of the 26th ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems (PODS 2007)*, pages 23–30. ACM, 2007.

[17] H. Pérez-Urbina, B. Motik, and I. Horrocks. Rewriting conjunctive queries under description logic constraints. In *Proc. of the Int. Workshop on Logics in Databases (LID 2008)*, 2008.

[18] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. on Data Semantics*, X:133–173, 2008.

[19] A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[20] M. Y. Vardi. Automata theory for database theoreticians. In *Proc. of the 8th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS 1989)*, pages 83–92. ACM, 1989.

[21] 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 *Proc. of the Workshop on Ontologies and Information Sharing*, pages 108–117, 2001.