Proc. 23rd Int. Workshop on Description Logics (DL2010), CEUR-WS 573, Waterloo, Canada, 2010.

Supporting the Development of Data Wrapping Ontologies (Extended Abstract)*

Lina Lubyte and Sergio Tessaris

KRDB Research Centre, Free University of Bozen-Bolzano

1 Introduction

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, such as information integration through mediated schemata [1], and the Semantic Web [2]. Ontologies provide a conceptual view of the application domain, which is closer to the user perspective, and automated reasoning can be leveraged to support exploration and querying of the underlying data sources.

In this paper we focus on the problem of designing ontologies which describe relational data sources, 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 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 newly introduced terms (concepts or roles) should be "backed" by data in the sources; i.e. queries over these terms should be rewritable w.r.t. data sources.

It is easy to provide examples where newly introduced terms will always return empty answers, regardless the actual data contained in the sources (see Section 3). This not necessarily because they are unsatisfiable in the usual model

* This paper is an excerpt from the ASWC 2009 paper "Supporting the Development of Data Wrapping Ontologies" by the same authors. The work presented in this paper has been partially funded by the European project ONTORULE.

theoretic meaning, but because there is no way of mapping them into the data sources.

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.

In [3] we introduced the problem and presented some preliminary results. The contribution of this paper is a generalisation of the these results, by providing algorithms to verify term emptiness for a more expressive class of ontology languages (see [4]). 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. Moreover we provide a technique to support the user in the "repair" of the empty terms and we present empirical study showing the benefits of our approach.

2 Preliminaries

To formalize ontologies, we use the DL $\mathcal{ELHI}$ [4]. For P an *atomic role*, an $\mathcal{ELHI}$ *basic role* has the form P or P^-. For A an *atomic concept*, an $\mathcal{ELHI}$ *basic concept* has the form $A, \exists R, \exists R.A$ or $B_1 \sqcap B_2$, where R is a basic role. An $\mathcal{ELHI}$ ontology is formalized in terms of a *TBox*, which is a set of *inclusion assertions* of the form $B_1 \sqsubseteq B_2$ or $R_1 \sqsubseteq R_2$, with B_1, B_2 basic concepts and R_1, R_2 basic roles. The actual data instances are instead stored in an *ABox*, that consists of a set of *membership assertions* of the form $A(a)$ or $R(a, b)$, with A an atomic concept, P an atomic role, and a, b constants[1]. An $\mathcal{ELHI}$ *knowledge base* (KB) $\mathcal{K}$ is a tuple $\langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox. We assume the "standard" DL semantics, with the *unique name assumption*.

A datalog rule is an expression of the form $\alpha(\boldsymbol{x}) \leftarrow \mathit{body}(\boldsymbol{x}, \boldsymbol{y})$, where $\alpha(\boldsymbol{x})$ is the *head* atom and $\mathit{body}(\boldsymbol{x}, \boldsymbol{y})$ is a set of *body* atoms. A *datalog program* Π is a set of datalog rules. The *extensional database (EDB) predicates* of Π are those that do not occur in the head atom of any rule in Π; all other predicates are called *intentional database (IDB) predicates*. A *datalog query* Q over an $\mathcal{ELHI}$ KB $\mathcal{K}$ is a tuple $\langle Q_\Pi, \Pi \rangle$, where Q_Π is a query predicate and Π is a datalog program whose predicates (except Q_Π) are concept and role names occurring in $\mathcal{K}$. Q is a *conjunctive query* (CQ) if Π contains exactly one rule with Q_Π as its head predicate not occurring in the body. A tuple of constants $\boldsymbol{a}$ is a *certain answer* to a datalog query Q over $\mathcal{K}$ iff $\mathcal{K} \cup \Pi \models Q_\Pi(\boldsymbol{a})$, where Π is considered to be a set of universally quantified implications with the usual first-order semantics. We use $\mathit{cert}(Q, \mathcal{K})$ to denote the set of all certain answers to Q over $\mathcal{K}$.

[1] As a matter of fact, an ABox is considered only *virtually*, while the actual data is stored in a relational DBMS and *wrapped* by means of an ontology; see [5].

3 Emptiness of Ontology Terms

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, 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.

To provide an intuition of the reasoning task let us consider a simple example depicted in Figure 1, where the bottom part represents the logical schema, the middle part the data source terms (connected with the relational sources by means of mappings, depicted with dashed arrows) and the top part the enriched fragment of the ontology. It is obvious that any query on Actor would always return empty answer, whatever the data sources may contain; while the concept represented by the same term would be satisfiable. The situation would be different if Actor was also restricted to elements whose range w.r.t. person_role was bound to ActingRole[2]. In this case, there could be instances of the database for which the same query on Actor would return a nonempty answer.

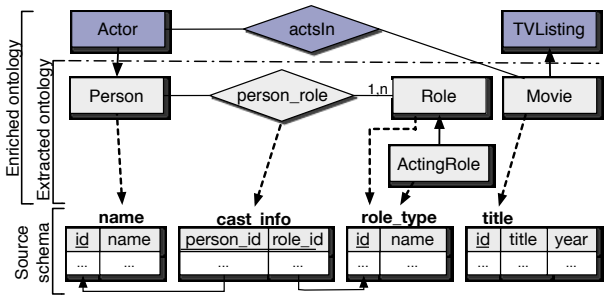

Fig. 1. Example of a simple data wrapper

Let $\Sigma_{\mathcal{DB}}$ denote the subset of terms occurring in $\mathcal{T}$ as "coming" from the data sources, i.e. data source terms. Given such $\Sigma_{\mathcal{DB}}$, a $\Sigma_{\mathcal{DB}}$-*ABox* is an ABox defined over $\Sigma_{\mathcal{DB}}$ only. Given a term η in $\mathcal{T}$, we call a *query for* η a CQ of the form $Q(x) \leftarrow \eta(x)$ (resp., $Q(x, y) \leftarrow \eta(x, y)$), for η an atomic concept (resp., an atomic role) in $\mathcal{T}$. Our goal is to test whether η is empty w.r.t. the data at the

[2] In DL terms this corresponds to an inclusion assertion $\exists$person_role.ActingRole $\sqsubseteq$ Actor.

sources, i.e., w.r.t. $\Sigma_{\mathcal{DB}}$. Clearly, such a test should involve the query answering process. That is, to verify emptiness of η, we have to check whether a query for η is empty given a TBox and a $\Sigma_{\mathcal{DB}}$-ABox.

Definition 1. *Let $\mathcal{T}$ be an $\mathcal{ELHI}$ TBox and η a term in $\mathcal{T}$ with query Q for η. Then, η* is empty w.r.t. *$\Sigma_{\mathcal{DB}}$ iff $cert(Q, \langle \mathcal{T}, \mathcal{A} \rangle) = \emptyset$ for every $\Sigma_{\mathcal{DB}}$-ABox $\mathcal{A}$.*

This defines the problem studied in this paper: given a term η in $\mathcal{T}$ with a CQ Q for η, test whether $cert(Q, \langle \mathcal{T}, \mathcal{A} \rangle) = \emptyset$ for every $\mathcal{A}$ whose assertions are over $\Sigma_{\mathcal{DB}}$ only. Note however that this does not imply that we will be necessarily computing $cert(Q, \langle \mathcal{T}, \mathcal{A} \rangle)$.

It is well known that the problem of computing certain answers in the presence of an incomplete database is often solved via *query rewriting* under constraints. Specifically, from [6] we have that given a conjunctive query Q over an $\mathcal{ELHI}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, we can compute another query Q', a *rewriting* of Q, such that the certain answers of Q over $\mathcal{K}$ and the answers of Q' over $\mathcal{A}$ only coincide, i.e., $cert(Q, \langle \mathcal{T}, \mathcal{A} \rangle) = cert(Q', \mathcal{A})$. Thus, we have the following:

Lemma 1. *Let $\mathcal{T}$ be an $\mathcal{ELHI}$ TBox and η a term in $\mathcal{T}$ with query Q for η. Let Q' be a rewriting of Q. Then, η* is empty w.r.t. *$\Sigma_{\mathcal{DB}}$ iff $cert(Q', \mathcal{A}) = \emptyset$ for every $\Sigma_{\mathcal{DB}}$-ABox $\mathcal{A}$.*

The above lemma 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 $\Sigma_{\mathcal{DB}}$-ABox. We will see later that for this purpose we will not need to compute the actual evaluation, however, we will employ the above relationship as described in the sequel.

4 Testing Emptiness

The rewriting of a CQ over $\mathcal{ELHI}$ KB is a datalog query [6]. Therefore, according to Lemma 1, our problem now comes down to testing emptiness of a query predicate in the rewritten datalog program. The problem of verifying emptiness of datalog predicates has been addressed by Vardi [7], showing that deciding emptiness of IDB predicates can be done 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, in turn, can be described by means of *expansion trees*. Importantly, [7] shows that we can get rid of variables when building expansion trees, obtaining *skeletons of expansion trees*. Then, an IDB predicate is empty in a datalog program, iff there is no skeleton tree for that predicate having as leaves EDB predicates only. We build our approach on the results of [7], and in particular on the possibility of building finitely labelled trees for IDB predicates.

For a term η with a CQ Q for η in an $\mathcal{ELHI}$ TBox $\mathcal{T}$, we devise our emptiness testing algorithm in four steps: *(i)* rewrite Q using procedure of [6], obtaining a datalog query $Q' = \langle Q_\Pi, \Pi \rangle$, *(ii)* add to Π auxiliary rules for making IDB

and EDB predicates explicit, *(iii)* for the resulting Datalog program with a query predicate Q_Π, build an AND-OR skeleton tree for Q_Π, and *(iv)* traverse the obtained tree by marking its nodes as empty/nonempty corresponding to empty/nonempty predicates, and, in turn, to empty/nonempty concepts and roles in $\mathcal{T}$. In the following we will elaborate on steps *(ii)-(iv)*; for details on the rewriting algorithm we refer to [6].

Given a datalog program Π with a query predicate Q_Π resulting from rewriting a CQ for a given term over $\mathcal{T}$, let Π^* denote a datalog program obtained by adding to Π rules of the form:

- $A(x) \leftarrow A(x)$, $P(x,y) \leftarrow P(x,y)$, for every predicate symbol $A, P \in \Pi$ corresponding to an atomic concept and role in $\mathcal{T}$, respectively, such that $A, P \notin \Sigma_{\mathcal{DB}}$ and A, P do not occur among the head atoms of any rule in Π;
- $A(x) \leftarrow A_{db}(x)$, $P(x,y) \leftarrow P_{db}(x,y)$ for every predicate symbol $A, P \in \Pi$ such that $A, P \in \Sigma_{\mathcal{DB}}$.

Note that an auxiliary rule $A(x) \leftarrow A(x)$ is equivalent to a tautology $A(x) \vee \neg A(x)$; thus, from a logical point of view, we do not change the semantics of Π.

The following definition describes the AND-OR skeleton tree that is associated to a datalog program (we assume all rules in Π^* are named).

Definition 2. *Given a datalog program Π^* and an IDB predicate Q_Π in Π^*, the associated* AND-OR skeleton tree for Q_Π in Π^*, *denoted* $\mathsf{tree}(Q_\Pi, \Pi^*)$, *is a labelled tree consisting of alternating levels of and-nodes and or-nodes such that*

- *the root of* $\mathsf{tree}(Q_\Pi, \Pi^*)$ *is a (and-)node labelled by* Q_Π,
- *for every and-node labelled by a predicate R in* $\mathsf{tree}(Q_\Pi, \Pi^*)$ *and for every rule r of Π^* having R as its head predicate, there exists a child* or-node *of R labelled by r,*
- *for every or-node labelled by a rule r in Π^*,* $\mathsf{tree}(Q_\Pi, \Pi^*)$ *has an* and-node *child for every atom g in the body of r, and the label of each such and-node is the predicate symbol of g.*

An and-node labelled by R in $\mathsf{tree}(Q_\Pi, \Pi^*)$ *is a leaf, if either* (i) *it is labelled with an EDB predicate,* (ii) *there are no rules in Π^* having R predicate in the head, or* (iii) *there is some other and-node in* $\mathsf{tree}(Q_\Pi, \Pi^*)$ *labelled by R that has already been expanded; we refer to such node as the* expanded equivalent of R, *denoted $eq(R)$.*

An or-subtree τ *in* $\mathsf{tree}(Q_\Pi, \Pi^*)$ *is a subtree of* $\mathsf{tree}(Q_\Pi, \Pi^*)$ *such that* (i) *for an and-node $R \in \tau$, τ contains one of the child or-nodes of R in* $\mathsf{tree}(Q_\Pi, \Pi^*)$, (ii) *for an or-node $r \in \tau$, τ contains all children of r in* $\mathsf{tree}(Q_\Pi, \Pi^*)$.

Example 1. Consider the data wrapping ontology from Figure 1. We list below the relevant axioms:

$$\mathsf{Movie} \sqsubseteq \mathsf{TVListing} \quad \exists \mathsf{actsIn}.\mathsf{Actor} \sqsubseteq \mathsf{Movie} \quad \mathsf{Actor} \sqsubseteq \mathsf{Person}$$

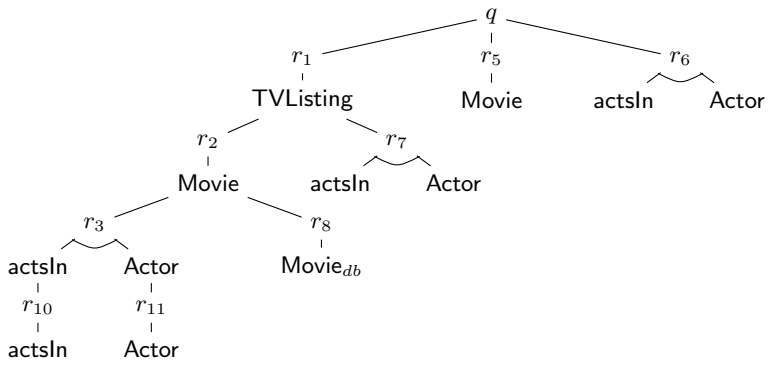

Fig. 2. Skeleton tree corresponding to the Datalog program of Example 1.

As can be seen from the figure, Movie and Person are linked to the data sources, i.e. they are data source terms. Suppose we want to test emptiness of TVListing term in the above ontology. The (partial) datalog program resulting from rewriting the query $q(x) \leftarrow \mathsf{TVListing}(x)$ is given below, together with auxiliary rules r_8 through r_{11} to make actsIn and Actor IDB predicates, and Person and Movie EDB predicates.

$r_1 : q(x) \leftarrow \mathsf{TVListing}(x)$
$r_2 : \mathsf{TVListing}(x) \leftarrow \mathsf{Movie}(x)$
$r_3 : \mathsf{Movie}(x) \leftarrow \mathsf{actsIn}(y, x), \mathsf{Actor}(y)$
$r_4 : \mathsf{Person}(x) \leftarrow \mathsf{Actor}(x)$
$r_5 : q(x) \leftarrow \mathsf{Movie}(x)$
$r_6 : q(x) \leftarrow \mathsf{actsIn}(y, x), \mathsf{Actor}(y)$
$r_7 : \mathsf{TVLising}(x) \leftarrow \mathsf{actsIn}(y, x), \mathsf{Actor}(y)$
$r_8 : \mathsf{Movie}(x) \leftarrow \mathsf{Movie}_{db}(x)$
$r_9 : \mathsf{Person}(x) \leftarrow \mathsf{Person}_{db}(x)$
$r_{10} : \mathsf{actsIn}(x, y) \leftarrow \mathsf{actsIn}(x, y)$
$r_{11} : \mathsf{Actor}(x) \leftarrow \mathsf{Actor}(x)$

The AND-OR skeleton tree for this datalog program is shown in Figure 2. Note that the children and-nodes of r_{10}, r_{11}, r_7, r_5 and r_6 are not further expanded, since they have isomorphic nodes that have already been expanded. The tree has 5 distinct or-subtrees, one of them e.g. formed from the path of or-nodes r_1, r_2 and r_8.

It is easy to show that each or-subtree of a given AND-OR skeleton tree corresponds to a skeleton of expansion tree defined in [7]. Therefore, a query predicate Q_Π is empty, iff all or-subtrees of $\mathsf{tree}(Q_\Pi, \Pi^*)$ are empty.

Definition 3. *Given an AND-OR skeleton tree* $\mathsf{tree}(Q_\Pi, \Pi^*)$ *for* Q_Π *in* Π^**, an* and-node R is empty in $\mathsf{tree}(Q_\Pi, \Pi^*)$ *if either* (i) *there is the expanded equivalent of* R*,* $eq(R)$*, that is empty in* $\mathsf{tree}(Q_\Pi, \Pi^*)$*;* (ii) R *is a leaf in* $\mathsf{tree}(Q_\Pi, \Pi^*)$*, it is not an EDB predicate, and there is no* $eq(R)$ *in* $\mathsf{tree}(Q_\Pi, \Pi^*)$*;* (iii) *all children or-nodes of* R *are empty. An* or-node r is empty in $\mathsf{tree}(Q_\Pi, \Pi^*)$ *if at least one child and-node of* r *is empty.*

The above definition provides the basis for a procedure for traversing a given AND-OR skeleton tree. While emptiness of Q_Π node can be decided by inspecting leaf nodes only, our algorithm traverses all the tree; this information will

be the main input for suggesting the "repairs" of empty terms, as described in Section 5. We illustrate this process with the following example.

Example 2 (Example 1 continued). We start with actsIn leaf, child of r_{10}, and mark it as empty (it is not an EDB predicate). This makes also its parent r_{10} and, in turn, actsIn and r_3 empty. To decide for Movie, we have to know emptiness of r_8. Movie_{db} is an EDB predicate, so it is nonempty. Consequently, we mark r_8 and Movie as nonempty, which determines non-emptiness for r_2 and then TVListing, r_1 and finally q. Actor leaf, child of r_{11}, is empty as well. Consequently, children of r_7 and r_6 are empty. In contrary, Movie, child of r_5 is marked as nonempty, because its expanded equivalent, child of r_2, is nonempty.

Indeed, we can construct a CQ $q(x) \leftarrow \mathsf{Movie}_{db}(x)$ from the AND-OR skeleton tree that witnesses non-emptiness for TVListing.

According to [6] and due to the fact that the input query for a given term has always single atom in its body, we have that the number of rules generated by the rewriting algorithm is exponential w.r.t. $\mathcal{T}$. Given n distinct IDB predicates in Π^*, the size of the AND-OR tree generated from Π^* is at most nm, where m is the maximum number of atoms in the body of a rule in Π^*. Thus, we have the following.

Theorem 1. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a $\mathcal{ELHI}$ KB, η a term in $\mathcal{T}$ and $\Sigma_{\mathcal{DB}}$ set of data source terms. Emptiness of η w.r.t. $\Sigma_{\mathcal{DB}}$ can be decided in time exponential in the size of $\mathcal{T}$.*

Note that the above result is optimal w.r.t. the complexity bounds from [8]: deciding emptiness of a term in $\mathcal{ELI}$ there is shown to be ExpTime-complete.

Finally, we stress the fact that, due to the rewriting algorithm [6], the technique presented in this section is applicable to ontology languages in the full spectrum of DLs from $\mathcal{ELHI}$ to *DL-Lite*$_{core}$ [9].

5 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 now present a method for supporting the repair of empty concepts and roles, consisting of a set of *repairing axioms* that can be seen as guidelines for ontology engineers.

To suggest a repair for an empty term, we naturally resort to the datalog program Π^* and the skeleton tree generated from Π^* by our emptiness testing algorithm. Indeed, the skeleton tree for a term η, by virtue of its construction, contains as nodes all and only *relevant* terms for η: 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 of its corresponding skeleton tree and to possibly *expand* those nodes by rendering them nonempty. The expansion should obviously be in correspondence with an addition or refinement of a term or/and assertion in the actual ontology.

Given an or-subtree τ in an AND-OR skeleton tree $\mathsf{tree}(Q_\Pi, \Pi^*)$ with all nodes marked, let $\Omega = [\omega_1, \ldots, \omega_n]$ denote the sequence of distinct sets of and-nodes[3] in τ, such that, intuitively, each ω_i contains a set of and-nodes that are empty in $\mathsf{tree}(Q_\Pi, \Pi^*)$ and are grouped in a bottom-up fashion by their depth. The next example illustrates this notion.

Example 3. Suppose rule r_8 was not present in the tree of Figure 4. Hence, TVListing is no longer nonempty. Ω defined above for the or-subtree following r_2, r_3 ancestors of TVListing is the following sequence: [{actsIn, Actor}, {Movie}, {TVListing}]. The intuition here is that, in order for TVLisintg to become nonempty, besides rendering TVListing itself nonempty, also Movie or both, actsIn and Actor, if rendered nonempty, would make TVListing nonempty as well. Instead the reason for a depth based ordering is that if both, actsIn and Actor were made nonempty, then the remaining terms in the sequence Movie and TVListing would become nonempty as well.

Thus, for each and-node R in ω, we consider R as a leaf in the tree and examine its possible expansions. In turn, to expand a leaf we 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. We exploit axioms in the ontology, rather than rules in the program, because, by virtue of [6], not all axioms are in one-to-one correspondence with rules in the computed rewriting.

For a node R corresponding to an atomic concept, say A, our repair service provides the following guidelines. First, it suggests to add an inclusion assertion with A on the right-hand side (line 6). This, from the modeling point of view, results in either defining role typing constraints (or domain and range) for a relationship defined by role P, if such is detected by means of mandatory participation constraints (and similarly if range restriction is given for P). Second, if $A \sqsubseteq B$ is present in $\mathcal{T}$ and B is nonempty, the user is warned with misplaced is-a relationship, i.e., possibly $B \sqsubseteq A$ should have been added instead of $A \sqsubseteq B$. Third, given $A \sqsubseteq B$ in $\mathcal{T}$ such that B has participation constraints to a nonempty role P, the algorithm suggests to assert participation constraints for A to P as well (and similarly if range restriction is known for P). Moreover, given a range concept, say C for P, if C is specialized by some concept D in the ontology, then the suggested axiom for A can also be specialized to D. Finally, the service suggests to assert A as a superclass of some concept B, and as a participating class to some role P, provided both A and P are known to be nonempty in $\mathcal{T}$. When $\mathcal{T}$ is small, such axioms could be included in the set of repairing axioms for every nonempty concept and role in $\mathcal{T}$. Otherwise, the task of selecting appropriate concepts and roles is left to the user.

If a given node R corresponds to a role, say P, the service generates axioms in a similar fashion. First, as before, it warns for misplaced role inclusions, provided such an axiom is present in $\mathcal{T}$. Then, if a root node being considered for repair

[3] Two and-nodes are considered distinct if their labels are distinct.

is a concept and not a role[4], then for every nonempty atomic concept A in $\mathcal{T}$ acting as a domain or range of P, the service suggests to add an axiom stating mandatory participation for A to the relationship defined by P (and the same for more specific concepts, as above). Finally, it hints to add an inclusion assertion between roles with P on the right-hand side, i.e. to make P more general than some role S that is nonempty in $\mathcal{T}$.

Note that the set of repairing axioms may also be empty, if there are no nonempty nodes in the tree that can be used for repair. In this case, we suggest to explicitly map to the sources either the actual empty term or any of its relevant terms.

Example 4 (Example 3 continued). Consider a data wrapping ontology from Figure 1 and suppose Movie is not mapped to the sources. To repair actsIn the user will be suggested to assert it as more general than person_role. This is obviously not meaningful, so there is no repair for actsIn. As for Actor, our repair service suggests the following axioms:

$$\mathsf{Person} \sqsubseteq \mathsf{Actor} \qquad \exists \mathsf{person\_role.Role} \sqsubseteq \mathsf{Actor}$$
$$\exists \mathsf{person\_role} \sqsubseteq \mathsf{Actor} \qquad \exists \mathsf{person\_role.ActingRole} \sqsubseteq \mathsf{Actor}$$

6 Evaluation

We have implemented services discussed in Sections 4 and 5 as a plug-in for Protégé 3.3[5] (we are in the process of porting them to Protégé 4) and evaluated their effectiveness with a usability study involving ten external users (see [5] for details).

We used showbiz domain for the study. In particular, for the sources, we used *IMDB* movie database, retrieved using IMDbPY[6]. The wrapping ontology, that we call showbiz, was obtained by first automatically extracting the bootstrap ontology from *IMDB* database together with mappings [10] (21 in total), and then by manually enriching it with terms and assertions to (partly) describe TV programmes. The obtained ontology contained 24 classes and 14 properties.

The subjects were randomly divided into two groups: five subjects without the support for testing emptiness of ontology terms and repairing them (group 1), and five subjects with the support of the above described plug-in (group 2). Then, each subject was given four simple queries over showbiz ontology but having empty answers: e.g. asking for all movies that have a genre, all TV listings and their kinds, etc. Given that, 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. The subjects in group 2 were additionally asked to fill in a questionnaire concerning their experience using the tool. The goal of this study

[4] While our procedure computes repairs for a contributive node, with a root node here we mean the node that one actually aims to repair, as e.g. TVListing in Example 3.

[5] `http://protege.stanford.edu`

[6] `http://imdbpy.sourceforge.net/`

was to compare the time taken and effort needed to complete the task between the two groups, and to evaluate user experience in using the plug-in.

The results of the study are promising. While the assertions added to an ontology in order to arrive to a solution were mostly correct and alike in both groups, the time taken to do it in group 2 was between 2-3 times less than in group 1. Specifically, the average time taken for group 1 was 39 minutes, and 20 minutes for group 2. 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. 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.

As mentioned, we have also collected user reactions to the tool. The questionnaire used for this purpose was composed of 10 short statements (e.g., "I found repair guidelines to be adequate"), 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. The key aspects, from the usability point of view, are that subjects in group 2 felt that they could effectively identify the reason for query emptiness using the tool (rated a total score of 19) and effectively repair empty terms using the tool (21 points), and strongly agreed that they could identify empty classes/properties and fix them using the tool faster than without it (25 points). Finally, the overall satisfaction of using the plug-in scores to 25.

7 Conclusions

This paper presents a technique for supporting ontology engineers in the development of ontologies for accessing relational data sources. We introduced the notion of emptiness of a given term 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). We have presented an optimal practical algorithm for deciding emptiness of terms in $\mathcal{ELHI}$ ontologies. Moreover, we have shown how the information generated by this algorithm can be exploited in order to support the engineer in "repairing" the ontology. The algorithm presented can be applied in other scenarios, e.g. for optimizing the rewriting by removing rules with empty predicates, or for guiding module extraction based on nonempty terms only (see [8]).

Recently there has appeared a contribution [8], carried out independently, that tackles a very similar problem but comes up in a different context. The authors study the computational complexity for the problem of predicate (and query) emptiness for a wide range of DLs. For the DLs $\mathcal{EL}$ and *DL-Lite*, they provide algorithms for verifying emptiness, taking a different approach from ours. While our algorithm is via translation to emptiness of IDB predicates in datalog, [8] instead uses reduction to standard ABox reasoning. Using their simple technique, emptiness of a term in $\mathcal{EL}$ can be decided in PTime. For $\mathcal{ELI}$, testing emptiness is shown to be already ExpTime-complete, by reduction to

subsumption/instance checking, but no algorithm is provided for this problem. As we mentioned, our practical algorithm is optimal w.r.t. the complexity bounds established there.

Levy [12] defined, in the context of datalog optimization, so-called *irrelevance claims* stating that a formula is irrelevant to a query w.r.t. a knowledge base and proposed algorithms for deciding irrelevance. However, this notion is rather different in nature from the emptiness problem we studied in this paper. In particular, it is a premise of a proof which may or may not be relevant to the deduction of a given formula. Therefore, those techniques cannot be directly applied.

Finally, we refer to the work in [13] as related, where, for queries having answers solely determined by the database predicates (the so-called DBox predicates with closed semantics, as apposed to the ABox), the authors show how to find a rewriting over such predicates. The restriction to determinacy may be however in some cases too strong, as for instance TVListing in Figure 1 is not *determined* by the database predicates but can be (in the classical setting) *rewritten* to a database term.

References

1. Lenzerini, M.: Data integration: A theoretical perspective. In: Proc. of PODS'02, ACM (2002) 233–346
2. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American (2001)
3. Lubyte, L., Tessaris, S.: Supporting the design of ontologies for data access. In: Workshop Notes of DL'08. (2008)
4. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proc. of IJCAI'05. (2005) 364–369
5. Lubyte, L., Tessaris, S.: Supporting the development of data wrapping ontologies. In: Proc. of ASWC'09. (2009) 31–45
6. Pérez-Urbina, H., Motik, B., Horrocks, I.: Tractable query answering and rewriting under description logic constraints. J. of Applied Logic (2009)
7. Vardi, M.Y.: Automata theory for database theoreticians. In: Proc. of PODS'89, ACM (1989) 83–92
8. Baader, F., Bienvenu, M., Lutz, C., Wolter, F.: Query and predicate emptiness in description logics. In: Proc. of KR'10. (2010) To appear.
9. Calvanese, D., Giacomo, G.D., Lembo, D., et al.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. J. of Automated Reasoning **39**(3) (2007) 385–429
10. Lubyte, L., Tessaris, S.: Automatic extraction of ontologies wrapping relational data sources. In Bhowmick, S., Küng, J., Wagner, R., eds.: Proc. of DEXA 2009. Volume LNCS 5690 of Springer. (2009) 128–142
11. Levy, A.Y.: Irrelevance Reasoning in Knowledge Based Systems. PhD thesis, Stanford University (1993)
12. Seylan, I., Franconi, E., de Bruijn, J.: Effective query rewriting with ontologies over DBoxes. In: Proc. of IJCAI'09. (2009) 923–930