High Performance Query Answering over DL-Lite Ontologies

Mariano Rodríguez-Muro and **Diego Calvanese**
KRDB Research Centre for Knowledge and Data
Free University of Bozen-Bolzano, Italy
{rodriguez,calvanese}@inf.unibz.it

Abstract

Current techniques for query answering over *DL-Lite* ontologies have severe limitations in practice, since they either produce complex queries that are inefficient during execution, or require expensive data pre-processing. In light of this, we present two complementary sets of results that aim at improving the overall peformance of query answering systems. We show how to create ABox repositories that are complete w.r.t. a significant portion of *DL-Lite* TBoxes, including those expressed in RDFS, but where the data is not explicitly expanded. Second, we show how to characterize ABox completeness by means of dependencies, and how to use these and equivalence to optimize *DL-Lite* TBoxes. These results allow us to reduce the cost of query rewriting, often dramatically, and to generate highly efficient queries. We have implemented a novel system for query answering over *DL-Lite* ontologies that incorporates these techniques, and we present a series of data-intensive evaluations that show their effectiveness.

1 Introduction

Current approaches to *ontology based query answering* (OBQA) with lightweight Description Logics (DLs) of the *DL-Lite* family [Calvanese et al., 2007] rely on query reformulation. These techniques are based on the idea of storing ABoxes in efficient data repositories, e.g., RDBMSs, and using the ontology's TBox to reformulate a given query into a new query that, when evaluated over the ABox repository, returns the certain answers to the original query. Experiments have shown that reformulations may be very large or complex, and that the execution of these suffers from poor performance. This triggered the development of alternative reformulation techniques [Pérez-Urbina, Motik, and Horrocks, 2010; Rosati and Almatelli, 2010; Kikot, Kontchakov, and Zakharyaschev, 2011], whose focus has been on the reduction of the number of generated queries/rules. These techniques produce reformulations that in many cases are polynomial in the size of the original query, however the worst-case complexity is still exponential.[1] Alternative approaches [Kontchakov et al., 2010] use the expansion of the ABox w.r.t. the intensional knowledge (i.e., the TBox) to avoid query reformulation almost entirely. This last approach offers good performance at query time, however, the cost of data expansion degrades initialization. We mention also the recent technique of [Gottlob and Schwentick, 2011], which produces worst-case polynomial rewritings at the cost of complicating their structure, so that we don't have evidence that such rewritings, though small, can be efficiently evaluated. In fact, we argue that a solution to the performance problem in OBQA systems requires a broader perspective of the problem, including, but beyond, the query reformulation technique.

Our first focus is on redundancy in reasoning, in particular due to *completeness* of the ABox w.r.t. the TBox. We say that an ABox $\mathcal{A}$ is complete w.r.t. a TBox axiom if the explicit data in the ABox satisfies the semantics of the axiom, for example, if $\mathcal{A}$ is complete with respect to $A \sqsubseteq B$, then for every a, if $A(a) \in \mathcal{A}$, then $B(a) \in \mathcal{A}$. One can see that being aware of the state of completeness of the ABox allows one to simplify the query answering process, and in some cases to avoid query reformulation altogether. In addition, explicitly expanding data is not the only way to complete an ABox in OBQA systems, e.g., the data in the ABox may already be (partially) complete or we could construct the ABox repository in such a way that it simulates completeness. We argue that with proper means to characterize, enforce and deal with completeness, we can optimize query answering.

Our second focus is the role of RDBMSs in OBQA systems. RDBMSs as ABox repositories have been proposed with the aim of exploiting their performance, however, the limitations of these systems have never been taken into account seriously. The most important limitation is that RDBMs excel only when faced with average case queries, or after careful tuning of the query and the system. In OBQA systems, in which queries are generated automatically, this is a crucial aspect, and it implies that the overall performance of query execution is not determined by the rewriting technique alone, but also by the ABox repository; in particular, by the choice of DB schema, the indexes defined over this schema, the mechanism that translates rewritten queries into SQL, and also the features and configuration of the RDBMS.

With these observations in mind, in this paper we approach the problem of efficient OBQA from two complementary directions. First we focus on completeness in query answering

[1] We consider here *DL-Lite*$_\mathcal{A}$, a variants of *DL-Lite* that allows for role inclusions.

systems and show that *(i)* (partial) completeness of ABoxes in OBQA systems that access external data sources (so called *OBDA systems*) is to be expected, and, more importantly, *(ii)* we can exploit the architecture of OBQA systems to produce *virtual ABoxes* that are complete for any TBox in the RDFS fragment of *DL-Lite*, without the need to generate new data. The consequence of this is that RDFS TBoxes in this fragment will become redundant and, for *DL-Lite*, TBox reasoning will only be necessary w.r.t. existential constants. We introduce one technique that aims at achieving completeness of ABoxes while allowing for the generation of efficient SQL queries without incurring in high-costs. Second, we present an approach to take into account completeness of the data with respect to *DL-Lite* TBoxes; we characterize completeness using *ABox dependencies* and show that it is possible to use dependencies to optimize a *DL-Lite*$_{\mathcal{A}}$ TBox in order to avoid redundancy in query answers, *independently of the adopted query reformulation technique*. The resulting TBoxes are optimal, in the sense that they will contain only those assertions that are necessary to guarantee correctness (i.e., completeness) of the computed certain answers. Additionally, we provide an ontology vocabulary simplification mechanism that reduces complexity of reasoning w.r.t. equivalent concepts and roles.

We have implemented a novel system for OBQA over *DL-Lite*$_{\mathcal{A}}$ ontologies that incorporates our techniques, and we present a series of evaluations and benchmarks that show that the techniques presented here are very effective in practice.

2 Preliminaries

We assume a fixed vocabulary V of *atomic concepts*, denoted A (possibly with subscripts), and *atomic roles*, denoted P, representing unary and binary relations, respectively, and an alphabet Γ of *(object) constants*.

Databases. We regard a *database (DB)* as a pair $\mathbf{D} = \langle \mathbf{R}, \mathbf{I} \rangle$, where $\mathbf{R}$ is a relational schema and $\mathbf{I}$ is an instance of $\mathbf{R}$. The active domain $\Gamma_{\mathbf{D}}$ of $\mathbf{D}$ is the set of constants appearing in $\mathbf{I}$, which we call *value constants*. An SQL query φ over a DB schema $\mathbf{R}$ is a mapping from a DB instance $\mathbf{I}$ of $\mathbf{R}$ to a set of tuples of value constants.

***DL-Lite* ontologies.** We introduce the DL *DL-Lite*$_{\mathcal{A}}$, on which we base our results. In *DL-Lite*$_{\mathcal{A}}$, a *basic role*, denoted R, is an expression of the form P or P^-, and a *basic concept*, denoted B, is an expression of the form A or $\exists R$. An *ontology* is a pair $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{T}$ is a TBox and $\mathcal{A}$ an ABox. A TBox is a finite set of

- *(positive) inclusions* $B_1 \sqsubseteq B_2$ or $R_1 \sqsubseteq R_2$,
- *disjointness assertions* $B_1 \sqsubseteq \neg B_2$, and
- *functionality assertions* $(\mathsf{funct}\ R)$.

An ABox is a finite set of *membership assertions* $A(c)$ or $P(c, c')$, where $c, c' \in \Gamma$. Moreover, *DL-Lite*$_{\mathcal{A}}$ imposes the syntactic restriction that a role P declared functional, via $(\mathsf{funct}\ P)$, or inverse functional, via $(\mathsf{funct}\ P^-)$, cannot be specialized, i.e., cannot appear in the right-hand side of a role inclusion assertion $R \sqsubseteq P$ or $R \sqsubseteq P^-$. We call *DL-Lite*$_{\mathtt{RDFS}}$ the fragment of *DL-Lite*$_{\mathcal{A}}$ in which we rule out concept inclusions of the form $B \sqsubseteq \exists R$, disjointness assertions, and functionality assertions.

Queries over ontologies. An *atom* is an expression of the form $A(t)$ or $P(t, t')$, where t and t' are *atom terms*, i.e., variables or constants in Γ. An atom is *ground* if it contains no variables. A *conjunctive query* (CQ) q over an ontology $\mathcal{O}$ is an expression of the form $q(\vec{x}) \leftarrow \beta(\vec{x}, \vec{y})$, where $\vec{x}$ is a tuple of distinct variables, called *distinguished*, $\vec{y}$ is a tuple of distinct variables not occurring in $\vec{x}$, called *non-distinguished*, and $\beta(\vec{x}, \vec{y})$ is a *conjunction* of atoms with variables in $\vec{x}$ and $\vec{y}$, whose predicates are atomic concepts and roles of $\mathcal{O}$. We call $q(\vec{x})$ the *head* of the query and $\beta(\vec{x}, \vec{y})$ its *body*. A *union of CQs* (UCQ) is a set of CQs (called disjuncts) with the same head. Given a CQ Q with body $\beta(\vec{z})$ and a tuple $\vec{v}$ of constants of the same arity as $\vec{z}$, we call a *ground instance* of Q the set $\beta[\vec{z}/\vec{v}]$ of ground atoms obtained by replacing in $\beta(\vec{z})$ each variable with the corresponding constant from $\vec{v}$.

Semantics. The semantics of *DL-Lite*$_{\mathcal{A}}$ is based as usual on the notion of FOL interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty *interpretation domain* and $\cdot^{\mathcal{I}}$ is an *interpretation function* (see, e.g., [Baader et al., 2003]). We just remark that we adopt the unique name assumption, which enforces that for each pair of constants c_1, c_2, if $c_1 \neq c_2$, then $c_1^{\mathcal{I}} \neq c_2^{\mathcal{I}}$. We also make use of the standard notions of *satisfaction*, *model*, and *entailment*.

Let $\Gamma_{\mathcal{A}}$ denote the set of constants appearing in an ABox $\mathcal{A}$. The *answer* to a CQ $Q = q(\vec{x}) \leftarrow \beta(\vec{x}, \vec{y})$ over $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ in an interpretation $\mathcal{I}$, denoted $ans(Q, \mathcal{O}, \mathcal{I})$, is the set of tuples $\vec{c} \in \Gamma_{\mathcal{A}} \times \cdots \times \Gamma_{\mathcal{A}}$ such that there exists a tuple $\vec{o} \in \Delta^{\mathcal{I}} \times \cdots \times \Delta^{\mathcal{I}}$ such that all facts in $\beta[(\vec{x}, \vec{y})/(\vec{c}^{\mathcal{I}}, \vec{o})]$ are true in $\mathcal{I}$, where $\vec{c}^{\mathcal{I}}$ denotes the tuple of objects that interpret the constants in $\vec{c}$. The answer to an UCQ Q in $\mathcal{I}$ is the union of the answers to all CQs in Q.

The *certain answers* to Q in $\mathcal{O}$, denoted $cert(Q, \mathcal{O})$, is the intersection of $ans(Q, \mathcal{O}, \mathcal{I})$ for each model $\mathcal{I}$ for $\mathcal{O}$. The *answer to Q over an ABox $\mathcal{A}$*, denoted $eval(Q, \mathcal{A})$, is the answer to Q over $\mathcal{A}$ viewed as a DB instance. A *perfect reformulation* of Q w.r.t. a TBox $\mathcal{T}$ is a query Q' such that for every ABox $\mathcal{A}$ such that $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable, $cert(Q, \langle \mathcal{T}, \mathcal{A} \rangle) = eval(Q', \mathcal{A})$.

Mappings. We adopt the definitions for ontologies with mappings from [Poggi et al., 2008], but we make some simplification for ease of presentation. In particular, in the following we assume that a database contains both value constants and object constants.[2] Given a TBox $\mathcal{T}$ and a DB $\mathbf{D}$, a *mapping (assertion)* m for $\mathcal{T}$ is an expression of the form $\varphi(\vec{x}) \rightsquigarrow \psi(\vec{t})$ where $\varphi(\vec{x})$ is an SQL query over $\mathbf{D}$

[2]Hence, we do not discuss here the use of Skolem terms in mappings, which allow one to bridge the impedance mismatch between a database storing values, and an ontology maintaining objects. We refer to [Poggi et al., 2008] for the details, and just observe that our techniques and the Quest system discussed in Section 5 fully support the general form of mappings that make use of Skolem terms.

with answer variables $\vec{x}$, and $\psi(\vec{t})$ is a CQ over $\mathcal{T}$ without non-distinguished variables. We call the mapping *simple* if the body of $\psi(\vec{t})$ consists of a single atom, and *complex* otherwise. A simple mapping *is for* an atomic concept A (resp., atomic role P) if the atom in the body of $\psi(\vec{t})$ has A (resp., P) as predicate symbol. In the following, we might abbreviate the query ψ in a mapping by showing only its body. A *virtual ABox* $\mathcal{V}$ is a tuple $\langle \mathbf{D}, \mathcal{M}\rangle$, where $\mathbf{D}$ is a DB and $\mathcal{M}$ a set of mappings, and an *ontology with mappings* is a tuple $\mathcal{OM} = \langle \mathcal{T}, \mathcal{V}\rangle$, where $\mathcal{T}$ is a TBox and $\mathcal{V} = \langle \mathbf{D}, \mathcal{M}\rangle$ is a virtual ABox in which $\mathcal{M}$ is a set of mappings for $\mathcal{T}$.

An interpretation $\mathcal{I}$ *satisfies* a mapping assertion $\varphi(\vec{x}) \rightsquigarrow \psi(\vec{t})$ w.r.t. a DB $\mathbf{D} = \langle \mathbf{R}, \mathbf{I}\rangle$ if for every tuple $\vec{v} \in \varphi(\mathbf{I})$ and for every ground atom X in $\psi[\vec{x}/\vec{v}]$ we have that: if X has the form $A(c)$, then $c^{\mathcal{I}} \in A^{\mathcal{I}}$, and if X has the form $P(c_1, c_2)$, then $(c_1^{\mathcal{I}}, c_2^{\mathcal{I}}) \in P^{\mathcal{I}}$. An interpretation $\mathcal{I}$ is a *model* of $\mathcal{V} = \langle \mathbf{D}, \mathcal{M}\rangle$, denoted $\mathcal{I} \models \mathcal{V}$, if it satisfies every mapping in $\mathcal{M}$ w.r.t. $\mathbf{D}$. A virtual ABox $\mathcal{V}$ *entails* an ABox assertion α, denoted $\mathcal{V} \models \alpha$, if every model of $\mathcal{V}$ is a model of α. $\mathcal{I}$ is a model of $\mathcal{OM} = \langle \mathcal{T}, \mathcal{V}\rangle$ if $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \models \mathcal{V}$. As usual, $\mathcal{OM}$ is *satisfiable* if it admits a model. We note that, in an ontology with mappings $\mathcal{OM} = \langle \mathcal{T}, \langle \mathbf{D}, \mathcal{V}\rangle\rangle$, we can always replace $\mathcal{M}$ by a set of *simple* mappings, while preserving the semantics of $\mathcal{OM}$. It suffices to *split* each complex mapping $\varphi \rightsquigarrow \psi$ into a set of simple mappings that share the same SQL query φ (see [Poggi et al., 2008]). In the following, we assume to deal only with simple mappings.

Dependencies. ABox dependencies are assertions that restrict the *syntactic* form of allowed ABoxes. In this paper, we focus on unary and binary inclusion dependencies only. A *unary (resp., binary) inclusion dependency* is an assertion of the form $B_1 \sqsubseteq_A B_2$, where B_1 and B_2 are basic concepts (resp., $R_1 \sqsubseteq_A R_2$, where R_1 and R_2 are basic roles). In the following, for a basic role R and constants c, c', $R(c, c')$ stands for $P(c, c')$ if $R = P$ and for $P(c', c)$ if $R = P^-$. An ABox $\mathcal{A}$ *satisfies* an inclusion dependency σ, denoted $\mathcal{A} \models \sigma$, if the following holds:

- if σ is $A_1 \sqsubseteq_A A_2$, then for all $A_1(c) \in \mathcal{A}$ we have $A_2(c) \in \mathcal{A}$;
- if σ is $\exists R \sqsubseteq_A A$, then for all $R(c, c') \in \mathcal{A}$ we have $A(c) \in \mathcal{A}$;
- if σ is $A \sqsubseteq_A \exists R$, then for all $A(c) \in \mathcal{A}$ there exists c' such that $R(c, c') \in \mathcal{A}$;
- if σ is $\exists R_1 \sqsubseteq_A \exists R_2$, then for all $R_1(c, c') \in \mathcal{A}$ there exists c'' such that $R_2(c, c'') \in \mathcal{A}$;
- if σ is $R_1 \sqsubseteq_A R_2$, then for all $R_1(c, c') \in \mathcal{A}$ we have $R_2(c, c') \in \mathcal{A}$.

An ABox $\mathcal{A}$ satisfies a set of dependencies Σ, denoted $\mathcal{A} \models \Sigma$, if $\mathcal{A} \models \sigma$ for each $\sigma \in \Sigma$. A set Σ of dependencies *entails* a dependency σ, denoted $\Sigma \models \sigma$, if for every ABox $\mathcal{A}$ s.t. $\mathcal{A} \models \Sigma$ we also have that $\mathcal{A} \models \sigma$. Given two queries Q_1, Q_2, we say that Q_1 *is contained in* Q_2 *relative to* Σ if $eval(Q_1, \mathcal{A}) \subseteq eval(Q_2, \mathcal{A})$ for each ABox $\mathcal{A}$ s.t. $\mathcal{A} \models \Sigma$.

3 Completeness in OBQA Systems

Completeness of the extensional information in the ABox w.r.t. the intensional information in the TBox plays an important role in OBQA. This notion can be formalized as follows.

Definition 3.1. Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}\rangle$ be a satisfiable *DL-Lite*$_{\mathcal{A}}$ ontology. We say that $\mathcal{A}$ is *complete for an assertion* $B_1 \sqsubseteq B_2$ in $\mathcal{T}$ if $\mathcal{A} \models B_1 \sqsubseteq_A B_2$, and that $\mathcal{A}$ is *complete for* $\mathcal{T}$ if it is complete for all assertions in $\mathcal{T}$. ■

In a *DL-Lite*$_{\mathcal{A}}$ ontology, the ABox is, in general, incomplete for the TBox. In fact, since *DL-Lite*$_{\mathcal{A}}$ does not enjoy the finite model property [Rosati, 2008], there are even TBoxes for which no (finite) complete ABox can exist. However, in practice, ABoxes may be already (partially) complete for a TBox. E.g., an ABox satisfying $A_1 \sqsubseteq_A A_2$ is complete for $A_1 \sqsubseteq A_2$.

The presence of completeness in an ABox for the inclusions in the TBox can greatly simplify the reasoning that needs to be carried out at query execution time. For example, the benefits of inducing completeness are well known in the context of RDFS/SPARQL query answering engines, in which *forward chaining* is used to completely avoid query reformulation at run-time. At the same time, expanding the data with respect to a TBox is not desirable, since this is a costly operation in terms of both space and time, even if done off-line. However, expanding data is not the only way to achieve completeness of an ABox, as we argue in the following.

3.1 Completeness in OBDA Systems

We have found evidence [Savo et al., 2010; Calvanese et al., 2010; Keet et al., 2008] that completeness is a common trait of OBDA systems, which are systems in which a virtual ABox $\langle \mathbf{D}, \mathcal{M}\rangle$ is defined from data in an *external* data source $\mathbf{D}$ through mappings $\mathcal{M}$. Note that when an external source is involved, mappings always exist, either as explicit logical assertions as the ones adopted in this paper, or because they are implicitly defined through application code. Mappings become crucial in determining the properties of the ABox. In particular, any dependencies that hold over the database, or any containment relationships that hold between the source queries used in the mappings will be reflected in the ABox as ABox dependencies.

Example 3.1. Let $\mathbf{R}$ be a DB schema with the relation schema *employee* with attributes *id*, *dept*, and *salary*, storing information about employees, their salaries, and the department they work for. Let $\mathcal{M}$ be the following mappings:

```
SELECT id,dept              ⇝  Employee(id) ∧
  FROM employee                WORKS-FOR(id,dept)

SELECT id,dept              ⇝  Manager(id) ∧
  FROM employee                MANAGES(id,dept)
  WHERE salary > 1000
```

where *Employee* and *Manager* are atomic concepts and *WORKS-FOR* and *MANAGES* are atomic roles. Then for *every* instance $\mathbf{I}$ of $\mathbf{R}$, we have that the ABox constructed from $\mathbf{R}$, $\mathbf{I}$, and $\mathcal{M}$ satisfies the following dependencies:

$$\begin{array}{rr} \mathit{Manager} \sqsubseteq_A \mathit{Employee} & \exists \mathit{MANAGES} \sqsubseteq_A \mathit{Manager} \\ \mathit{Manager} \sqsubseteq_A \exists \mathit{MANAGES} & \exists \mathit{WORKS\text{-}FOR} \sqsubseteq_A \mathit{Employee} \\ \mathit{Employee} \sqsubseteq_A \exists \mathit{WORKS\text{-}FOR} & \end{array}$$

In particular, the first dependency follows from the containment between the two SQL queries used in the mappings, and the remaining dependencies follow from the fact that we populate *WORKS-FOR* (resp., *MANAGES*) using the same SQL query used to populate *Employee* (resp., *Manager*). ■

The strong relationship between ABox completeness and the mappings of OBDA systems emerges when we turn our attention to the semantics of the data sources and the objective of building an ABox from an existing source. First, we note that any given DB is based on some conceptual model. At the same time, if we associate the data of any given DB to the concepts and roles of a TBox $\mathcal{T}$ to build an ABox, it follows that this data is *semantically related* to these concepts and roles, and that the conceptual model of the DB has some common aspects with the semantics of $\mathcal{T}$. It is precisely these common aspects that get manifested as dependencies between queries in the mappings and that give rise to completeness in ABoxes. Therefore, the degree of completeness of an ABox in an OBDA system is in direct relation with the similarity between the semantics of the DB's conceptual model and that of the TBox, and with the degree in which the DB itself complies to the conceptual model that was used to design it. To illustrate these observations, we extend Example 3.1.

Example 3.2. Note that the intended meaning of the data stored in $\mathbf{D}$ is as follows: *(i)* employees with a salary higher than 1,000 are managers, *(ii)* managers manage the department in which they are employed, and *(iii)* every employee works for a department. This meaning is reflected in the mapping assertions of $\mathcal{M}$, and every ABox generated from $\mathbf{D}$ through $\mathcal{M}$ will be (partially) complete for a TBox $\mathcal{T}$ that shares part of the semantics of $\mathbf{D}$. In other words, such a TBox will present redundancy. For example, if $\mathcal{T}$ is

$$\begin{array}{rcl@{\qquad}rcl} \mathit{Manager} & \sqsubseteq & \mathit{Employee} & \exists \mathit{MANAGES}^- & \sqsubseteq & \mathit{Department} \\ \mathit{Manager} & \sqsubseteq & \exists \mathit{MANAGES} & \exists \mathit{WORKS\text{-}FOR}^- & \sqsubseteq & \mathit{Department} \\ \mathit{Employee} & \sqsubseteq & \exists \mathit{WORKS\text{-}FOR} & & & \end{array}$$

then the first column is redundant w.r.t. Σ. Instead, the semantics of the second column is not captured by the mappings. In such an OBDA system, we should reason only w.r.t. *Department*. This can be accomplished by optimizing $\mathcal{T}$ w.r.t. Σ using the technique presented in Section 4. ■

Completeness of an ABox introduces *redundancy* in reasoning. It is well known in automated theorem proving that by addressing and exploiting redundancy, the runtime of an inference algorithm may drop from exponential to polynomial [Gottlob and Fermüller, 1993; Joyner, 1976]. We claim that this is also the case for OBQA, where not only dependencies may already hold in ABoxes, but where we can induce them in efficient ways, as shown in the following.

3.2 Inducing Dependencies

We consider now OBQA systems that rely on query rewriting, and on a RDBMS to store the ABox data and to evaluate the rewritten query over the ABox. In order to store the ABox, such systems define a relational schema, and mechanisms to populate and query the corresponding database by referring to the concepts and roles of the TBox; in other words, the ABox is managed transparently to the user. We can formalize this setting through the notion of *virtual ABox* $\mathcal{V} = \langle\langle \mathbf{R}, \mathbf{I}\rangle, \mathcal{M}\rangle$, as presented in Section 2, where the form of the database schema $\mathbf{R}$ and of the mappings $\mathcal{M}$ may vary, depending on the actual implementation chosen for the ABox repository. We model such an OBQA system as an ontology with mappings, $\mathcal{O} = \langle \mathcal{T}, \mathcal{V}\rangle$, which allows us to analyze the properties of ABox repositories, or the result of operations over them, by abstracting away from actual implementation details of the repository itself.

With respect to completeness, this allows us to generalize the idea of ABox expansion to the concept of *dependency induction technique*. We call *dependency induction technique* a procedure that, given an ontology with mappings $\mathcal{O} = \langle \mathcal{T}, \mathcal{V}\rangle$, uses $\mathcal{T}$ to compute an alternative virtual ABox $\mathcal{V}'$ s.t. the number of inclusions in $\mathcal{T}$ for which $\mathcal{V}'$ is complete is higher than those for $\mathcal{V}$. *ABox expansion* and *forward-chaining* are procedures that modify the data in $\mathbf{I}$ for a fixed DB schema $\mathbf{R}$. However, we could also improve completeness of $\mathcal{V}$ by manipulating the structure of $\mathbf{R}$, the way we initially construct $\mathbf{I}$, or the mappings $\mathcal{M}$.

The critical aspect in dependency induction for OBQA systems is the trade-off between the degree of completeness being induced and the system's performance/cost for data loading and querying. For example, a system that resorts to inference materialization[3] will in general offer a good performance at query time. However, the cost of loading the data in such systems is increased and the system will require additional space to store the inferences. In scenarios in which reasoning is non-trivial due to the size of the ontologies, the additional loading time may be in the range of days, and the additional data may increase the space requirements of the application from gigabytes to terabytes [LePendu et al., 2010].

We present now a dependency induction mechanism that provides good trade-offs on all these aspects and that does not generate new data in $\mathbf{I}$. Notably, OBQA systems with virtual ABoxes created using this procedure will be complete for *DL-Lite*$_{\mathtt{RDFS}}$ TBoxes, thus making such TBoxes redundant. In the case of *DL-Lite*$_{\mathcal{A}}$ TBoxes, the only non-redundant inclusion assertions will be those involving existentially quantified individuals.

Semantic Index for OBQA systems. The basic idea of the semantic index technique is to encode the implied *is-a* relationships of $\mathcal{T}$ in the values of *numeric indexes* that we assign to concept and role names. ABox membership assertions are then inserted in the DB using these numeric values. The aim is to construct a virtual ABox for the system in which the mappings can retrieve most of the implied instances of any concept or role by posing simple range queries to the DB, which are queries that can be evaluated efficiently in modern DBMSs using B-tree indexes. Our proposal is strongly related to techniques for managing large transitive relations in knowledge bases [Agrawal, Borgida, and Jagadish, 1989], however, our interest is not in the hierarchies themselves, as in previous work, but in querying instance data that is associated to those hierarchies. It is also related to well known

[3]This is the traditional way of dealing with RDFS and OWL inferences in RDF triple stores.

techniques for XPath query evaluation [DeHaan et al., 2003]. Formally, a semantic index is defined as follows.

Definition 3.2. Given a *DL-Lite$_\mathcal{A}$* TBox $\mathcal{T}$ and its vocabulary V, *a semantic index for* $\mathcal{T}$ is a pair of mappings $\langle idx, range\rangle$ with $idx : V \to \mathbb{N}$ and $range : V \to 2^{\mathbb{N}\times\mathbb{N}}$, such that, for each pair E_1, E_2 of atomic concepts or atomic roles in V, we have that $\mathcal{T} \models E_1 \sqsubseteq E_2$ iff there is a pair $\langle \ell, h\rangle \in range(E_2)$ such that $\ell \leq idx(E_1) \leq h$. ■

Using a semantic index $\langle idx, range\rangle$ for a TBox $\mathcal{T}$, we construct a virtual ABox $\mathcal{V} = \langle\langle \mathbf{R}, \mathbf{I}\rangle, \mathcal{M}\rangle$ with the completeness properties described above by proceeding as follows. We define the DB schema $\mathbf{R}$ with a universal-like relation $T_C[\mathtt{c1}, \mathtt{idx}]$ for storing ABox concept assertions, and a relation $T_R[\mathtt{c1}, \mathtt{c2}, \mathtt{idx}]$ for storing ABox role assertions, such that `c1` and `c2` have type *constant* and `idx` has type *numeric*. Given an ABox $\mathcal{A}$, we construct $\mathbf{I}$ such that for each $A(c) \in \mathcal{A}$ we have $\langle c, idx(A)\rangle \in T_C$, and for each $P(c, c') \in \mathcal{A}$ we have $\langle c, c', idx(P)\rangle \in T_R$. The schema and the index allow us to define, for each atomic concept A and each atomic role P, a set of range queries over $\mathbf{D}$ that retrieve most constants c, c' such that $\mathcal{O} \models A(c)$ or $\mathcal{O} \models P(c, c')$. For example, if $range(A) = \{\langle 2, 35\rangle\}$, we define 'SELECT c1 FROM T_C WHERE idx >= 2 AND idx <= 35'. We use these queries to define the mappings in $\mathcal{M}$ as follows[4]:

- for each atomic concept A and each $\langle \ell, h\rangle \in range(A)$, we add the mapping $\sigma_{\ell\leq idx\leq h}(T_C) \rightsquigarrow A(c_1)$;
- for each atomic role P and each $\langle \ell, h\rangle \in range(P)$, we add the mapping $\sigma_{\ell\leq idx\leq h}(T_R) \rightsquigarrow P(c_1, c_2)$;
- for each pair of atomic roles P, P' such that $\mathcal{T} \models P'^{-} \sqsubseteq P$ and each $\langle \ell, h\rangle \in range(P')$ we add the mapping $\sigma_{\ell\leq idx\leq h}(T_R) \rightsquigarrow P(c_2, c_1)$;
- for each atomic concept A, each atomic role P such that $\mathcal{T} \models \exists P \sqsubseteq A$ (resp., $\exists P^{-} \sqsubseteq A$), and each $\langle \ell, h\rangle \in range(P)$, we add the mapping $\sigma_{\ell\leq idx\leq h}(T_R) \rightsquigarrow A(c_1)$ (resp., $\sigma_{\ell\leq idx\leq h}(T_R) \rightsquigarrow A(c_2)$);
- last, we replace any pair of mappings $\sigma_{\ell\leq idx\leq h}(T_C) \rightsquigarrow A(c_1)$ and $\sigma_{\ell'\leq idx\leq h'}(T_C) \rightsquigarrow A(c_1)$ such that $\ell' \leq h$ and $\ell \leq h'$ by the mapping $\sigma_{\min(\ell,\ell')\leq idx\leq \max(h,h')}(T_C) \rightsquigarrow A(c_1)$ (similarly for role mappings).

A semantic index could be trivially constructed by assigning to each atomic concept and role a unique (arbitrary) value and a set of ranges that covers all the values of its subsumees. However, this is not effective for optimizing query answering since the size of the semantic index (specifically, the number of pairs in $range(A)$ and $range(P)$) determines the size of $\mathcal{M}$, and this in turn determines exponentially the size of the final SQL query. To avoid an exponential blow-up, we create $\langle idx, range\rangle$ using the implied hierarchies as follows.

Let $\mathcal{T}$ be a TBox, and D_C the minimal DAG that represents the *implied is-a* relation between all atomic concepts of $\mathcal{T}$ (i.e., the *transitive reduct* of the concept hierarchy)[5]. Then we can construct idx by initializing a counter $i = 0$, and visiting the nodes in D_C in a depth-first fashion starting from the root nodes. At each step and given the node N visited at that step, if $idx(N)$ is undefined, set $idx(N) = i$ and $i = i + 1$, else if $idx(N)$ is defined, backtrack until the next node for which idx is undefined. Now, to generate $range$ we visit the nodes in D_C starting from the leafs and going up. For each node N in the visit, if N is a leaf in D_C, then we set $range(N) = \{\langle idx(N), idx(N)\rangle\}$, and if N is not a leaf, then we set $range(N) = merge(\{\langle idx(N), idx(N)\rangle\} \cup \bigcup_{N_i \mid N_i\to N\in D_C} range(N_i))$, where $merge$ is a function that, given a set r of ranges, returns the minimal set r' of ranges that has coverage equal to r, e.g., $merge(\{\langle 5,7\rangle, \langle 3,5\rangle, \langle 9,10\rangle\}) = \{\langle 3,7\rangle, \langle 9,10\rangle\}$. We proceed exactly in the same way with the DAG D_R representing the role hierarchy.

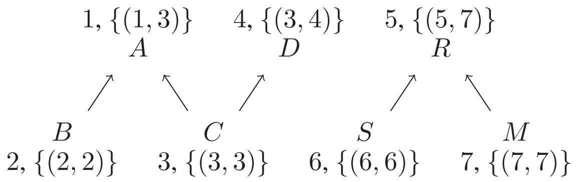
$$\begin{tikzcd}
 & \overset{1,\{(1,3)\}}{A} & & \overset{4,\{(3,4)\}}{D} & & \overset{5,\{(5,7)\}}{R} & \\
\underset{2,\{(2,2)\}}{B} \arrow[ur] & & \underset{3,\{(3,3)\}}{C} \arrow[ul, dash] \arrow[ur] & & \underset{6,\{(6,6)\}}{S} \arrow[ur] & & \underset{7,\{(7,7)\}}{M} \arrow[ul, dash]
\end{tikzcd}$$

Figure 1: D_C, D_R, and the values for *idx* and *range*.

$$\begin{array}{ll}
\sigma_{1\leq idx\leq 3}(T_C) \rightsquigarrow A(c_1) & \sigma_{5\leq idx\leq 7}(T_R) \rightsquigarrow R(c_1, c_2) \\
\sigma_{2\leq idx\leq 2}(T_C) \rightsquigarrow B(c_1) & \sigma_{6\leq idx\leq 6}(T_R) \rightsquigarrow S(c_1, c_2) \\
\sigma_{3\leq idx\leq 3}(T_C) \rightsquigarrow C(c_1) & \sigma_{7\leq idx\leq 7}(T_R) \rightsquigarrow M(c_1, c_2) \\
\sigma_{3\leq idx\leq 4}(T_C) \rightsquigarrow D(c_1) & \sigma_{5\leq idx\leq 7}(T_R) \rightsquigarrow D(c_1)
\end{array}$$

Figure 2: The mappings created by the technique.

$$\begin{array}{rclcrcl}
B & \sqsubseteq_A & A & \qquad & C & \sqsubseteq_A & A \\
C & \sqsubseteq_A & D & & \exists R & \sqsubseteq_A & D \\
S & \sqsubseteq_A & R & & M & \sqsubseteq_A & R
\end{array}$$

Figure 3: The induced dependencies.

Example 3.3. Let A, B, C, D be atomic concepts, let R, S, M be atomic roles, and consider the TBox $\mathcal{T} = \{B \sqsubseteq A, C \sqsubseteq A, C \sqsubseteq D, D \sqsubseteq \exists R, \exists R \sqsubseteq D, S \sqsubseteq R, M \sqsubseteq R\}$. Let the DAGs D_C and D_R for $\mathcal{T}$ be the ones depicted in Figure 1. The technique generates idx and $range$ as indicated in Figure 1, generates the mappings in Figure 2, and for any ABox, the technique generates a virtual ABox $\mathcal{V}$ that satisfies all the dependencies Σ in Figure 3. Then, we will have that in query answering, rewriting is only necessary w.r.t. $D \sqsubseteq \exists R$. We will present in Section 4 a technique to optimize the TBox to take this into account. ■

We make some remarks about the effectiveness of this technique. The factor that defines performance of the semantic index is the number of ranges for each atomic concept

[4]Here we use relational algebra expressions instead of SQL to simplify the exposition.

[5]We assume that $\mathcal{T}$ does not contain a cyclic chain of basic concept or role inclusions. This can be done w.l.o.g. since any TBox containing such cyclic chain can be transformed into one where the chain has been removed by using the technique described at the end of Section 4.

or role. This highly depends on the order of the edges and nodes in D_C. Determining the optimal DAG for a semantic index is a complex problem for which the work presented in [Agrawal, Borgida, and Jagadish, 1989] might provide guide. However, our experiments show that in practice even non-optimal semantic indexes behave well (see Section 5).

4 Optimizing TBoxes

Dependencies. Information about completeness of a (virtual) ABox, in the form of dependencies, can be used to avoid redundancy during reasoning. For example, during query reformulation we can use conjunctive query containment (CQC) with respect to dependencies to avoid the generation of redundant queries. However, this approach is expensive, since CQC is an NP-complete problem (even ignoring dependencies), and such optimizations would need to be performed every time a query is reformulated. We show now how we can improve efficiency by pre-processing the TBox before performing query reformulation. In particular, given a TBox $\mathcal{T}$ and a set Σ of dependencies, we show how to compute a TBox $\mathcal{T}'$ s.t. the set $\{\alpha \mid \mathcal{T}' \models \alpha\}$ is in general smaller than the set $\{\alpha \mid \mathcal{T} \models \alpha\}$ and for every query Q the certain answers are preserved if Q is executed over an ABox that satisfies Σ. Specifically, our goal is to determine when an inclusion assertion of $\mathcal{T}$ is *redundant w.r.t.* Σ. To do so we use the following auxiliary notions.

Definition 4.1. Let $\mathcal{T}$ be a TBox, B, C basic concepts, R, S basic roles, and Σ a set of dependencies over $\mathcal{T}$. A *$\mathcal{T}$-chain from B to C in $\mathcal{T}$* (resp., a *Σ-chain from B to C in Σ*) is a sequence of inclusion assertions $(B_i \sqsubseteq B'_i)_{i=0}^n$ in $\mathcal{T}$ (resp., a sequence of inclusion dependencies $(B_i \sqsubseteq_A B'_i)_{i=0}^n$ in Σ), for some $n \geq 0$, such that: $B_0 = B$, $B'_n = C$, and for $1 \leq i \leq n$, we have that B'_{i-1} and B_i are basic concepts s.t., either *(i)* $B'_{i-1} = B_i$, or *(ii)* $B'_{i-1} = \exists R'$ and $B_i = \exists R'^-$, for some basic role R'. A $\mathcal{T}$-chain from R to S in $\mathcal{T}$ (resp., a Σ-chain from R to S in Σ) is a sequence of inclusion assertions $(R_i \sqsubseteq R'_i)_{i=0}^n$ in $\mathcal{T}$ (resp., a sequence of inclusion dependencies $(R_i \sqsubseteq_A R'_i)_{i=0}^n$ in Σ), for some $n \geq 0$, such that: $R_0 = R$, $R'_n = S$ and for $1 \leq i \leq n$, we have that $R'_{i-1} = R_i$. ■

Intuitively, when there is a $\mathcal{T}$-chain from B to C, the existence of an instance of B in a model of $\mathcal{T}$ implies the existence of an instance of C. For a Σ-chain, this holds for ABox assertions. We use $\mathcal{T}$-chains and Σ-chains to characterize redundancy as follows.

Definition 4.2. Let $\mathcal{T}$ be a TBox, B, C basic concepts, R, S basic roles, and Σ a set of dependencies. The inclusion assertion $B \sqsubseteq C$ is *directly redundant in $\mathcal{T}$ w.r.t.* Σ if *(i)* $\Sigma \models B \sqsubseteq_A C$ and *(ii)* for every $\mathcal{T}$-chain $(B_i \sqsubseteq B'_i)_{i=0}^n$ with $B'_n = B$ in $\mathcal{T}$, there is a Σ-chain $(B_i \sqsubseteq_A B'_i)_{i=0}^n$. The inclusion assertion $R \sqsubseteq S$ is *directly redundant in $\mathcal{T}$ w.r.t.* Σ if *(i)* $\Sigma \models R \sqsubseteq_A S$, *(ii)* for every $\mathcal{T}$-chain $(B_i \sqsubseteq B'_i)_{i=0}^n$ with $B'_n = \exists R$, there is a Σ-chain $(B_i \sqsubseteq_A B'_i)_{i=0}^n$, and *(iii)* for every $\mathcal{T}$-chain $(R_i \sqsubseteq R'_i)_{i=0}^m$ with $R'_m = R$, there is a Σ-chain $(R_i \sqsubseteq_A R'_i)_{i=0}^m$. Then, $B \sqsubseteq C$ (resp., $R \sqsubseteq S$) is *redundant in $\mathcal{T}$ w.r.t.* Σ if *(a)* it is directly redundant, or *(b)* there exists $B' \neq B$ (resp., $R' \neq R$) s.t. *(i)* $\mathcal{T} \models B' \sqsubseteq C$ (resp., $\mathcal{T} \models R' \sqsubseteq S$), *(ii)* $B' \sqsubseteq C$ (resp., $R' \sqsubseteq S$) is not directly redundant in $\mathcal{T}$ w.r.t. Σ, and *(iii)* $B \sqsubseteq B'$ (resp., $R \sqsubseteq R'$) is directly redundant. ■

Given a TBox $\mathcal{T}$ and a set of dependencies Σ, we apply our notion of redundancy w.r.t. Σ to the assertions in $\mathcal{T}$ to obtain a TBox $\mathcal{T}'$ that is equivalent to $\mathcal{T}$ for certain answer computation. It turns out that we have to take into account also implied assertions and dependencies. We denote with $sat(\mathcal{T})$ the *saturation* of $\mathcal{T}$, i.e., the set of *DL-Lite$_\mathcal{A}$* assertions α s.t. $\mathcal{T} \models \alpha$. Notice that $sat(\mathcal{T})$ is finite, hence, it is a valid TBox. Similarly, $sat(\Sigma)$ denotes the *saturation* of Σ, i.e., the set of dependencies σ s.t. $\Sigma \models \sigma$.

Definition 4.3. Given a TBox $\mathcal{T}$ and a set of dependencies Σ over $\mathcal{T}$, the *optimized version of $\mathcal{T}$ w.r.t.* Σ, denoted $\mathcal{T}_\Sigma^{opt}$, is the set of inclusion assertions $\{\alpha \in sat(\mathcal{T}) \mid \alpha$ is not redundant in $sat(\mathcal{T})$ w.r.t. $sat(\Sigma)\}$. ■

Correctness of using $\mathcal{T}_\Sigma^{opt}$ instead of $\mathcal{T}$ when computing certain answers follows from the following theorem.

Theorem 4.1. *Let $\mathcal{T}$ be a TBox and Σ a set of dependencies over $\mathcal{T}$. Then for every ABox $\mathcal{A}$ such that $\mathcal{A} \models \Sigma$ and every UCQ Q over $\mathcal{T}$, we have that $cert(Q, \langle \mathcal{T}, \mathcal{A} \rangle) = cert(Q, \langle \mathcal{T}_\Sigma^{opt}, \mathcal{A} \rangle)$.*

Proof. First, we note that during query answering, only the positive inclusions are relevant, hence we ignore disjointness and functionality assertions. Since $sat(\mathcal{T})$ adds to $\mathcal{T}$ only entailed assertions, $cert(Q, \langle \mathcal{T}, \mathcal{A} \rangle) = cert(Q, \langle sat(\mathcal{T}), \mathcal{A} \rangle)$, for every Q and $\mathcal{A}$, and we can assume w.l.o.g. that $\mathcal{T} = sat(\mathcal{T})$. Moreover, $cert(Q, \langle \mathcal{T}, \mathcal{A} \rangle)$ is equal to the evaluation of Q over $chase(\mathcal{T}, \mathcal{A})$. (We refer to [Calvanese et al., 2007] for the definition of chase for a *DL-Lite$_\mathcal{A}$* ontology.) Hence it suffices to show that for every $B \sqsubseteq C$ that is redundant with respect to Σ, $chase(\mathcal{T}, \mathcal{A}) = chase(\mathcal{T} \setminus \{B \sqsubseteq C\}, \mathcal{A})$, similarly for $R \sqsubseteq S$. We give the proof for $B \sqsubseteq C$; the one for $R \sqsubseteq S$ is similar. Specifically, we prove that if $B \sqsubseteq C$ is redundant (hence, removed in $\mathcal{T}_\Sigma^{opt}$), then there is always a $chase(\mathcal{T}, \mathcal{A})$ in which $B \sqsubseteq C$ is never applicable. Assume by contradiction that $B \sqsubseteq C$ is applicable to some assertion $B(c)$ during some step in $chase(\mathcal{T}, \mathcal{A})$. We distinguish two cases that correspond to the cases of Definition 4.2.

(a) Case where $B \sqsubseteq C$ is directly redundant, and hence $\Sigma \models B \sqsubseteq_A C$. We distinguish two subcases: *(i)* $B(c) \in \mathcal{A}$. Since $\mathcal{A} \models B \sqsubseteq_A C$, we have $C(c) \in \mathcal{A}$, and hence $B \sqsubseteq C$ is not applicable to $B(c)$. Contradiction. *(ii)* $B(c) \notin \mathcal{A}$. Then there is a sequence of chase steps starting from some ABox assertion $B'(c')$ that generates $B(c)$. Such a sequence requires a $\mathcal{T}$-chain $(B_i \sqsubseteq B'_i)_{i=0}^n$ with $B_0 = B'$ and $B'_n = B$, such that each $B_i \sqsubseteq B'_i$ is applicable in $chase(\mathcal{T}, \mathcal{A})$. Then, by the second condition of direct redundancy, there is a Σ-chain $(B_i \sqsubseteq_A B'_i)_{i=0}^n$. Since $\mathcal{A} \models B_0 \sqsubseteq_A B'_0$ we have that $B'_0(c') \in \mathcal{A}$ and hence $B_0 \sqsubseteq B'_0$ is not applicable to $B'(c')$. Contradiction.

(b) Case where $B \sqsubseteq C$ has been removed by Definition 4.2(*b*), and hence there exists $B' \neq B$ such that $\mathcal{T} \models B' \sqsubseteq C$. First we note that any two oblivious chase sequences for $\mathcal{T}$ and $\mathcal{A}$ produce results that are equivalent w.r.t. query answering. Then it is enough to show that there exists

some $chase(\mathcal{T}, \mathcal{A})$ in which $B' \sqsubseteq C$ is always applied before $B \sqsubseteq C$ and in which $B \sqsubseteq C$ is never applicable. Again, we distinguish two subcases: *(i)* $B(c) \in \mathcal{A}$. Then, since $B \sqsubseteq B'$ is directly redundant, we have that $\Sigma \models B \sqsubseteq_A B'$. Since $\mathcal{A} \models \Sigma$, we have that $B'(c) \in \mathcal{A}$, and given that $B' \sqsubseteq C$ is always applied before $B \sqsubseteq C$, $C(c)$ is added to $chase(\mathcal{T}, \mathcal{A})$ before the application of $B \sqsubseteq C$, hence $B \sqsubseteq C$ is in fact not applicable. Contradiction. *(ii)* $B(c) \notin \mathcal{A}$. Then, arguing as in Case *(a.ii)*, using $B \sqsubseteq B'$ instead of $B \sqsubseteq C$, we can derive a contradiction. □

Optimality. A critical aspect of computing an optimized TBox is ensuring that all inferences triggered by this TBox are non-redundant. That is, every assertion in the optimized TBox is necessary to preserve the certain answer for every query Q and ABox $\mathcal{A}$. Optimality of $\mathcal{T}_\Sigma^{opt}$ is provided by the following theorem.

Theorem 4.2. *Let α be a TBox assertion such that $\mathcal{T}_\Sigma^{opt} \models \alpha$, and let $\mathcal{T}'$ be a TBox such that $\mathcal{T}' \subseteq \mathcal{T}_\Sigma^{opt}$ and $\mathcal{T}' \not\models \alpha$. Then there is a CQ q over $\mathcal{T}$ and an ABox $\mathcal{A}_0$ such that $\mathcal{A}_0 \models \Sigma$ and $cert(q, \langle \mathcal{T}_\Sigma^{opt} \rangle, \mathcal{A}_0) \neq cert(q, \langle \mathcal{T}', \mathcal{A}_0 \rangle)$.*

Proof. We give the proof for the case when α is an inclusion $B \sqsubseteq C$ between basic concepts. The proof for the case when α is a role inclusion is similar.

Since $\mathcal{T}_\Sigma^{opt} \models \alpha$ we have that $\mathcal{T} \models \alpha$ and α is not redundant in $\mathcal{T}$ w.r.t. Σ. Hence, neither *(a)* nor *(b)* of Definition 4.2 hold for α. Then, due to *(a)*, either *(i)* $\Sigma \not\models B \sqsubseteq_A C$ or *(ii)* there is some $\mathcal{T}$-chain $(B_i \sqsubseteq B_i')_{i=0}^n$ with $B_n' = B$ s.t. there is no corresponding Σ-chain $(B_i \sqsubseteq_A B_i')_{i=0}^n$.

In Case *(i)*, consider a minimal ABox $\mathcal{A}_0$ s.t. $\mathcal{A}_0 \models \Sigma$ and $B(a) \in \mathcal{A}_0$, for some constant a. Since $\Sigma \not\models B \sqsubseteq_A C$, we can construct $\mathcal{A}_0$ such that $C(a) \notin \mathcal{A}_0$. For the CQ $q() \leftarrow C(x)$, we have that $cert(q, \langle \mathcal{T}_\Sigma^{opt} \rangle, \mathcal{A}_0) = true$. We have to show that $cert(q, \langle \mathcal{T}', \mathcal{A}_0 \rangle) = false$. Assume by contradiction that this does not hold. Then there must be a basic concept B' (possibly C itself) s.t. *(1)* $\mathcal{T}' \models B' \sqsubseteq C$, *(2)* $B' \sqsubseteq C$ is not directly redundant in $\mathcal{T}'$ w.r.t. Σ, and *(3)* $B \sqsubseteq B'$ is directly redundant in $\mathcal{T}'$ w.r.t. Σ. Indeed, Condition *(3)* is necessary to ensure that $B'(a') \in \mathcal{A}_0$, for some constant a', while Conditions *(1)* and *(2)* are needed to ensure that $\langle \mathcal{T}', \mathcal{A}_0 \rangle \models C(a')$. But this contradicts the initial assumption that $B \sqsubseteq C$ is not redundant in $\mathcal{T}$ w.r.t. Σ.

In Case *(ii)*, consider a $\mathcal{T}$-chain $(B_i \sqsubseteq B_i')_{i=0}^n$ with $B_n' = B$ s.t. there is no corresponding Σ-chain $(B_i \sqsubseteq_A B_i')_{i=0}^n$. Consider a minimal ABox $\mathcal{A}_0$ s.t. $\mathcal{A}_0 \models \Sigma$ and $B_0(a_0) \in \mathcal{A}_0$, and take again the CQ $q() \leftarrow C(x)$. Due to the $\mathcal{T}$-chain and since $\mathcal{T}_\Sigma^{opt} \models B \sqsubseteq C$, we have that $cert(q, \langle \mathcal{T}_\Sigma^{opt}, \mathcal{A}_0 \rangle) = true$. We have to show again that $cert(q, \langle \mathcal{T}', \mathcal{A}_0 \rangle) = false$. The $\mathcal{T}$-chain ensures that $\langle \mathcal{T}', \mathcal{A}_0 \rangle \models B(a_0)$. Then we can reason as in Case *(i)*. □

Complexity and implementation. The described technique has also the nice property that $\mathcal{T}_\Sigma^{opt}$ can be computed in polynomial time in the size of $\mathcal{T}$. This is possible because the computation of direct redundancy is reducible to checks over ancestors and descendants over DAGs that represent reachability in the *is-a* hierarchy of $\mathcal{T}$ and reachability in $\mathcal{T}$-chains. We now present the construction of these DAGs.

Definition 4.4. Given a TBox $\mathcal{T}$, an *is-a DAG for* $\mathcal{T}$, denoted $D(\mathcal{T})$, is a DAG formed by a set V of nodes and a set E of edges that satisfies the following conditions:

- for each basic concept B (resp., basic role R), there is a node $n(B)$ (resp., $n(R)$) in V;
- for each assertion $B_1 \sqsubseteq B_2 \in \mathcal{T}$, there is an edge $n(B_1) \to n(B_2)$ in E;
- for each $R_1 \sqsubseteq R_2 \in \mathcal{T}$, there are edges $n(R_1) \to n(R_2)$, $n(\exists R_1) \to n(\exists R_2)$, and $n(\exists R_1^-) \to n(\exists R_2^-)$ in E.

Given a set Σ of dependencies, we define the *is-a DAG* for Σ, denoted $D(\Sigma)$, analogously, by using Σ instead of $\mathcal{T}$, and $\sqsubseteq_A$ instead of $\sqsubseteq$. ■

Intuitively, the reachability relations of the nodes in an *is-a* DAG for $\mathcal{T}$ or Σ allow us to represent succinctly all the inclusion assertions in $sat(\mathcal{T})$ or $sat(\Sigma)$. We use these DAGs to compute all implications of $\mathcal{T}$ and Σ. Likewise, we can use DAGs to represent the chains that exist in $sat(\mathcal{T})$ and $sat(\Sigma)$ as follows.

Definition 4.5. Given a TBox $\mathcal{T}$ (resp., a set Σ of dependencies), a *chain DAG for* $\mathcal{T}$, denoted $D^c(\mathcal{T})$ (resp., *chain DAG for* Σ, denoted $D^c(\Sigma)$), is an isa-DAG for $\mathcal{T}$ extended s.t. for each R and each B, if $n(B) \to n(\exists R) \in E$ then $n(B) \to n(\exists R^-) \in E$ and, if $n(B) \to n(\exists R^-) \in E$ then $n(B) \to n(\exists R) \in E$. ■

By making use of these DAGs, we can efficiently construct $\mathcal{T}_\Sigma^{opt}$. We omit the construction due to space constraints.

Theorem 4.3. *Given a TBox $\mathcal{T}$ and a set Σ of dependencies, $\mathcal{T}_\Sigma^{opt}$ can be constructed in polynomial time in the size of $\mathcal{T}$ and of Σ.*

Proof sketch. First note that we can pre-process a DAG in polynomial time to obtain (effectively, through a hash-map) constant time access to all descendants and ancestors of a node in the DAG. It is easy to see that using the DAGs $D(\mathcal{T})$, $D(\Sigma)$, $D^c(\mathcal{T})$, and $D^c(\Sigma)$ as described in Definitions 4.4 and 4.5, one can compare in polynomial time the chains in $\mathcal{T}$ and Σ for any given B by comparing the descendants of B in the DAGs. Then, for any given inclusion assertion α, we can check redundancy in polynomial time. Since the number of assertions to check is quadratic, the whole procedure runs in polynomial time. □

Last, note that reducing redundancy computation to descendants/ancestors checks in DAGs, gives us a direct way to implement the method presented here.

Dealing with equivalences. We discuss now a conceptually very simple form of optimization that, however, has a strong effect on performance of OBQA approaches based on query rewriting, namely the optimization of the vocabulary of $\mathcal{T}$ and $\mathcal{A}$ w.r.t. concept an role equivalences. The idea behind this optimization is the following: given two atomic concepts C_1 and C_2 such that $\mathcal{T} \models C_1 \equiv C_2$, we choose one of the two concepts, say C_1, and replace with C_1 every reference to C_2 in $\mathcal{T}$ and $\mathcal{A}$; also, during query answering, we replace every atom referring to C_2 with an equivalent atom referring to C_1 (similarly for roles and inverse roles). This simplification allows us to reduce the number of inferences done at query answering time and, given the exponential nature of many reasoning algorithms, can considerably increase performance.

Note also that $\mathcal{T} \models C_1 \equiv C_2$ if and only if $\mathcal{T}$ contains a set $\{B_1 \sqsubseteq B_2, \ldots, B_{n-1} \sqsubseteq B_n, B_n \sqsubseteq B_1\}$ of inclusions between basic concepts, with $B_1 = C_1$ and $B_n = C_2$, i.e., a cycle between C_1 and C_2. Hence, this technique allows us to transform $\mathcal{T}$ intro a cycle-free TBox, i.e., a DAG, that can be used to compute all entailments of $\mathcal{T}$ as long as we keep track of the vocabulary simplification we performed.

5 Evaluation

We now present two evaluations of the techniques presented here, more details can be found at https://babbage.inf.unibz.it/trac/obdapublic/wiki/Semanticindex.

LUBM3X. This evaluation tests the effect of the optimizations presented in this paper on the performance of query rewriting engines. For this evaluation we developed LUBM3X-lite, a variation of the LUBM ontology benchmark [Guo, Pan, and Heflin, 2005] that differs from the original in two ways: *(i)* LUBM3X-lite consists of 3 *copies* of each entity and each axiom of the original LUBM ontology. In these copies we appended the suffixes 1X, 2X and 3X to the original entity names. Last we added axioms of the form $A1X \sqsubseteq A2X$ and $A2X \sqsubseteq A3X$ for each concept A in the original LUBM (resp., for each role). This extension allowed us to deal with the fact that LUBM is rather flat, and the number of inferences required for each query is often low; *(ii)* The expressivity of LUBM goes slightly beyond that of OWL 2 QL. To deal with this, we approximated the ontology to OWL 2 QL expressivity while keeping as much of the original entailments as possible. We replaced all axioms of the form $A \equiv B \sqcap \exists R.C$ with $A \sqsubseteq B$ and $A \sqsubseteq \exists R.C$, removed all axioms of the form $\mathtt{transitive}(R)$ and added the axioms *GradStudent* $\sqsubseteq$ *Student*, *Director* $\sqsubseteq$ *Employee*, and *ResearchAssistant* $\sqsubseteq$ *Employee* (which are entailed by LUBM); *(iii)* Each SPARQL query of LUBM was modified to refer to the 3X concepts and roles. The main feature of LUBM3X-lite is that it increases the computational cost of answering queries with LUBM while it allows us to reuse the queries and data generators of LUBM. LUBM3X-lite can be downloaded from our site.

Rewriting performance. To evaluate rewriting performance we used 3 systems: Quest 1.6.1, a new system developed by us that implements all the aforementioned techniques, Requiem [Pérez-Urbina, Motik, and Horrocks, 2010] and Presto [Rosati and Almatelli, 2010]. These systems were chosen because each implements one of the two classes of rewriting techniques proposed up to now. Quest and Requiem use techniques that manipulate CQs during rewriting, while Presto manipulates non-recursive Datalog programs (DPs).

Q	Qs(b)	Qs(f)	Rq(n)	Rq(g)	Pr(d)	Pr(c)
1	441	348	24	**12**	451	31
2	78185	609	19881	198219	**60**	1730
3	18	33	**10**	44	28	35
4	111	176	274	570	**31**	35
5	1342	1137	1080	3418	**35**	37
6	2	3	1	1	1	1
7	1979	53	411	1324	**33**	46
8	4950	47	256	1160	**18**	42544
9	-	1466	-	-	**45**	245
10	8495	5	4	**1**	28	28
11	3	2	1	1	17	14
12	154	**5**	41	54	19	52
13	457	270	210	1011	34	**29**
14	1	1	0	0	1	0

Table 1: Reformulation time (ms), fastest in bold

Q	Qs(b)	Qs(f)	Rq(n)	Rq(g)	Pr(d)	Pr(c)
1	9	**1**	42	3	4	3
2	17496	**1**	17496	972	22	729
3	153	**1**	150	3	4	3
4	162	**1**	1356	27	28	27
5	1476	**1**	5322	69	67	66
6	12	**1**	12	12	13	12
7	1620	**1**	1512	108	19	108
8	3888	**1**	1296	1296	31	2916
9	-	**1**	-	-	22	324
10	36	**1**	111	12	13	12
11	18	**1**	18	18	10	18
12	162	**1**	162	162	16	162
13	1845	**1**	1560	15	16	15
14	3	**1**	3	3	4	3

Table 2: Reformulation size (# CQs/Rules), smallest in bold

We can find other examples of CQ-based implementations in QuOnto [Calvanese et al., 2007], Owlgres [Stocker and Smith, 2008] and Rapid [Chortaras, Trivela, and Stamou, 2011]. The time required for each of the systems to compute the perfect reformulation is shown in Table 1, while the number of CQs/rules in the output is shown in Table 2. The symbol '-' indicates a timeout after 200s. These tests were performed on an OS X machine with a 2x2.66 Ghz Core i7 processor and 6 GB of RAM dedicated to the Java VM.

First we note that the performance of Quest's *bare* reformulation engine (col. Qs(b)), i.e., without the optimizations mentioned before nor with CQC pruning of the output, is comparable in performance to that of Requiem in naive mode (col. Rq(n)), w.r.t. both rewriting time and size of the output. Moreover, both engines have a much lower performance than Presto in DP mode (col. Pr(d)). However, once the techniques presented in this paper are activated (col. Qs(f)), the rewriting engine of Quest outperforms Requiem's fully optimized mode (col. Rq(g)) and becomes close in performance to Presto. Note that Quest's reformulation engine did not change, only the input TBox and the dependencies available for CQC pruning. This confirms that the optimizations proposed in this paper do have a very strong impact on the efficiency of reformulation engines.

With respect to the size of the output, the deep hierarchies of LUBM3x-lite force Requiem to generate tens or thousands of CQs while Presto's DP-based rewritings contain tens of rules. Note that when Presto expands those DPs into UCQs (col. Pr(c)), their size expands considerably. At the same time, Quest always generates one single CQ as rewriting. The reason for Quest's small rewritings is that the optimized TBox entails few axioms of the form $A \sqsubseteq \exists R$ and some role inclusions related to implications of the form $A \sqsubseteq \exists R.B$. This allows Quest to generate very few rewritings, which in addition get further pruned using CQC w.r.t. the dependencies generated by the semantic index. With respect to the SQL generated by those rewritings, this aspect is not defined in Requiem or Presto. In the case of Quest, all these queries generate either one single SQL query with very few nested unions, or unions of very few select-project-join (SPJ) queries. This is so because in Quest disjunction is mostly addressed by the numeric ranges of the semantic index instead of union operations.

Execution performance. Requiem and Presto cannot be used for query execution because they lack an ABox layer. To evaluate this aspect of query answering, we used Owlgres 0.1 and Sesame 2.6. Owlgres is a system that relies on query rewriting (UCQ-based). Sesame is an RDF Java framework that includes its own implementation of a *native* triple store, i.e., a specialized form of database that is optimized for fast execution of SPARQL queries. Sesame supports RDFS reasoning by means of forward-chaining (i.e., inference materialization). Quest and Requiem used PostgreSQL 9.1 for ABox storage. The dataset consisted of a total of 50 "universities" (unis), i.e., 6863227 ABox assertions (≈2.3Gb in size) generated using LUBM's original data generator. The data set is divided in 50 unis of approx. 137000 assertions each.

We used the same data loading mechanism for all systems, i.e., first we loaded the LUBM3x-lite TBox, then we loaded each of the 50 dataset one at a time. Sesame's loading speed was initially ≈40s/uni, and as more data was loaded, the time increased up to ≈250s/uni, for a total time of 2hrs and 20m. Owlgres loading speed was constant at ≈ 107s/uni, for a total of 1hr and 25m. In the case of Quest, loading speed was constant at ≈3.3s/uni, for a total of 104s for the entire dataset with additional 220s for the creation of indexes. The reason for this rather pronounced gap in performance in favor of Quest is because Quest loads exactly as many rows as there are in the input. Moreover, the process does not consider existing data. In contrast, Sesame's forward chaining algorithm generates many times more assertions than there are in the input; moreover, the application of the forward chaining rules requires querying the already inserted data, an operation that grows in complexity as the database grows. Owlgres also queries the existing data, apparently to perform consistency checking and to generate URI identifiers. We also suspect that the loading mechanism of Owlgres is not very optimized, using SQL `INSERT` statements instead of Postgres `COPY` command as Quest does.

The performance of query execution of the systems is shown in Table 3, where (c) indicates a cold run, i.e., done after a full system reboot to clear the DB and OS caches, and (w) indicates the mean of 5 *warm* runs respectively. For cold runs, where the core query answering performance is shown, Quest outperforms Owlgres and Sesame on average, due to

Q	Qs(c)	Ow(c)	Se(c)	Qs(w)	Ow(w)	Se(w)
1	1504	1756	**653**	4	9	**1**
2	**134061**	218423	246373	30931	4361	**4012**
3	1312	**81**	185	1086	7	**1**
4	462	897	**251**	373	68	**2**
5	6888	4770	**260**	3275	942	**16**
6	**1905**	19926	23979	**1568**	8062	4765
7	563	232	**176**	367	90	**3**
8	**2644**	11048	2692	923	10577	**275**
9	**29807**	49535	165731	22557	**11282**	14300
10	7	16	**2**	5	14	**1**
11	444	54	**33**	360	13	**1**
12	136	334	**50**	14	324	**1**
13	3170	**11**	16	3131	9	**1**
14	**1135**	6862	13856	**1132**	7184	3593
T	**184038**	313947	454257	65727	42945	**26970**

Table 3: Execution time (ms), fastest in bold.

some 'hard' queries dominating the results. For warm runs, where OS disk caches and DB caches are heavily involved, Sesame outperforms the rest. Also note that Sesame performs better in simple queries, but in general, is outperformed by Quest and Owlgres (i.e., by Postgres) in complex queries.

Nevertheless, we now comment on some of the DB aspects of both systems. Owlgres and Quest define one table for each kind of ABox assertions, i.e., concept, role (object property), and attribute (data properties) assertions, however, Owlgres defines multi-column B+tree indexes for some tables, while Quest only defines single-column indexes; we believe this could be a factor in Quest's slightly lower performance for warm runs. We also observed that in queries 2 and 9, Postgres query plans for Quest's SQL queries incur in severe errors. In particular, the estimation of the row-output of several query-plan operations is much lower than the actual output. Smaller errors of this kind also appear in other queries, and in some of Owlgres queries. We believe that the reason for these wrong estimates is wrong or poor table/index statistics, and the performance of both systems should improve once the DB's configuration is improved.

Last, comparing the performance of the SQL query rewriting techniques of Owlgres and Quest is difficult due to several factors. First, we found that Owlgres' rewriting engine is incomplete in that it does not consider entailments about existential constants, which trivializes the query rewriting process and in turn, the final SQL. Second, Owlgres implements an optimization technique that detects CQ emptiness due to concept/role emptiness (which is tracked at load time). This has a strong effect with our benchmark since the LUBM data generator does not generate data for concepts and roles of the LUBM3x extended vocabulary. To conclude, we believe that in order to reach any conclusions w.r.t. the SQL rewriting techniques of both systems, it is necessary to further study their performance using a better RDBMS engine and better benchmark data.

The next evaluation covers some of these points, i.e., uses a commercial DBMS engine that should produce better query plans, and uses data from a real application.

Resource Index. We now describe an evaluation of the cost of the Semantic Index as well as the performance of the queries it generates in the presence of very large volumes of data. This experiment was carried out using data from the *Resource Index* (RI) [LePendu et al., 2010] application, winner of the Semantic Web Challenge Open Track 2010. The RI is an application that offers semantic search services over 22 well known collections of biomedical documents (i.e., resources). The semantics of the search is defined by the hierarchical information of $\approx$ 200 ontologies, including well known bio-medical ontologies like the Gene Ontology, SNOMEDCT, NCI Thesaurus, etc. The workflow of the RI can be summarized as follows: *(i)* Using natural language processing, each document is analyzed and annotated with one or more concepts from the ontologies. The annotations can be seen as ABox assertions, e.g., *Cervical_Cancer*$('\mathtt{doc224}')$. *(ii)* Users pose queries of the form $q(x) \leftarrow A_1(x) \wedge \cdots \wedge A_n(x)$, where each A_i is a concept from one of the ontologies in the collection. To compute the answers to the queries, the RI executes the original query over the expanded ABox data. The ontologies in the RI's amount to $\approx$3 million concepts and $\approx$2.5 million *sub-class* assertions. The annotation process generates a very large volume of data. For the resource used in this experiment, the *Clinical Trials.gov* (CT) collection, the annotation process generates $\approx$181 million ABox assertions (i.e., data triples), corresponding to $\approx$14 GB of data. The CT resource is only one out of 22 resources managed by the RI. The total amount of expanded data managed by the RI is $\approx$1.5 TB.

Our experimentation focused on the cost of query answering in this high-end scenario. We explored *(a)* query reformulation as unions of SPJ queries (CNF), or SPJ queries with nested views (DNF), *(b)* query answering using a Semantic Index, and *(c)* query answering with ABox expansion. The experiment used all the *subClass* assertions from the RI's ontologies and all the annotations for the CT resource. Note that given the limited expressivity of the TBox used by this application, we can avoid query reformulation w.r.t. the TBox by storing data using a Semantic Index. We stored the data in a DB2 9.7 DB hosted in a Linux virtual machine with 4x2.67 Ghz Intel Xeon processors (only one core was used) and 4 GB of RAM available to DB2. We issued several queries, the one we describe here is $q(x) \leftarrow$ *DNA_Repair_Gene*$(x) \wedge$*Antigen_Gene*$(x) \wedge$*Cancer_Gene*(x). The selectivity of the query is high, returning a total of 2 distinct resources. The performance of each technique is as follows: *(a)* when rewriting w.r.t. $\mathcal{T}$ in CNF form the result is one SQL query with 467874 disjuncts, when rewriting in DNF (as UCQ-based rewriters do), the result is a union of 467874 SPJ queries; none of these queries is executable by DB2 with our system setup; *(b)* when we rewrite the query using the Semantic Index technique, the result is a single SQL query involving 3 range disjunctions; the query requires 3.582s to execute (0.082s if the DB is warm, e.g., the indexes have been preloaded); the time required to compute the semantic index is 27s; the size of the semantic index $\approx$4 GB; *(c)* if the ABox is expanded and we execute the original query, the execution requires 3 s (0 s if warm). With respect to the cost of the expansion, LePendu et al. indicate that a straightforward expansion of the CT resource requires $\approx$7 days, and generates $\approx$140 GB of data and, after a careful optimization of the process (including data partitioning, parallelization, etc.) this time can be reduced to $\approx$40 minutes. Given these results, we believe that the Semantic Index is possibly a better option than data expansion, due to the drastic cost of the latter. Moreover, it scales to dimensions in which pure query reformulation may be impossible.

6 Conclusions and Future Work

In this paper we focused on issues of performance query answering with ontologies. Several directions can be taken in the future. First, the Semantic Index technique presented in Section 3 needs to be extended in order to support TBox updates in an efficient way; such operation may require a re-assignment of (none, some, or all of) the already computed indexes, which in turn may require updating the ABox assertions stored in the database; given the possibly high cost of such operation, it is important to devise a technique in which these updates are minimal and efficient. In the context of optimizations w.r.t. completeness can be also applied in the more strict 'virtual' OBDA context, in which data sources are independent and data can only be accessed 'on-the-fly'. An initial exploration of this problem can be found in [Rodríguez-Muro and Calvanese, 2011]. Moreover, although TBox pre-processing is the best choice to start optimizing w.r.t. completeness, it is also a necessary step during query reformulation. With respect to evaluation, our experiments, although very positive, are preliminary and further experimentation is required. In particular, it is necessary to further study indexing and configuration of DBMSs for the queries generated by the semantic index.

Acknowledgements

We thank Dr. Paea LePendu for providing us with the RI data and supporting us in understanding this application. We thank Sergejs Pugacs for his work on the implementation and evaluation of the Semantic Index. This research has been partially supported by the EU under the ICT Collaborative Project ACSI (Artifact-Centric Service Interoperation), grant n. FP7-257593.

References

Agrawal, R.; Borgida, A.; and Jagadish, H. V. 1989. Efficient management of transitive relationships in large data and knowledge bases. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 253–262.

Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; and Patel-Schneider, P. F., eds. 2003. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press.

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

Calvanese, D.; Keet, C. M.; Nutt, W.; Rodriguez-Muro, M.; and Stefanoni, G. 2010. Web-based graphical querying of databases through an ontology: the WONDER system. In *Proc. of the 25th ACM Symposium on Applied Computing (SAC 2010)*, 1388–1395.

Chortaras, A.; Trivela, D.; and Stamou, G. B. 2011. Optimized query rewriting for OWL 2 QL. In *Proc. of the 23rd Int. Conf. on Automated Deduction (CADE 2011)*, 192–206.

DeHaan, D.; Toman, D.; Consens, M. P.; and Özsu, M. T. 2003. A comprehensive XQuery to SQL translation using dynamic interval encoding. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 623–634.

Gottlob, G., and Fermüller, C. G. 1993. Removing redundancy from a clause. *Artificial Intelligence* 61(2):263–289.

Gottlob, G., and Schwentick, T. 2011. Rewriting ontological queries into small nonrecursive Datalog programs. In *Proc. of the 24th Int. Workshop on Description Logic (DL 2011)*.

Guo, Y.; Pan, Z.; and Heflin, J. 2005. LUBM: A benchmark for OWL knowledge base systems. *J. of Web Semantics* 3(2–3):158–182.

Joyner, W. H. 1976. Resolution strategies as decision procedures. *J. of the ACM* 23(3):398–417.

Keet, C. M.; Alberts, R.; Gerber, A.; and Chimamiwa, G. 2008. Enhancing web portals with Ontology-Based Data Access: the case study of South Africa's Accessibility Portal for people with disabilities. In *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008)*, volume 432 of *CEUR Electronic Workshop Proceedings,* http://ceur-ws.org/.

Kikot, S.; Kontchakov, R.; and Zakharyaschev, M. 2011. On (in)tractability of OBDA with OWL 2 QL. In *Proc. of the 24th Int. Workshop on Description Logic (DL 2011)*.

Kontchakov, R.; Lutz, C.; Toman, D.; Wolter, F.; and Zakharyaschev, M. 2010. The combined approach to query answering in *DL-Lite*. In *Proc. of the 12th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2010)*, 247–257.

LePendu, P.; Noy, N.; Jonquet, C.; Alexander, P.; Shah, N.; and Musen, M. 2010. Optimize first, buy later: Analyzing metrics to ramp-up very large knowledge bases. In *Proc. of the 9th Int. Semantic Web Conf. (ISWC 2010)*, volume 6496 of *Lecture Notes in Computer Science*. Springer. 486–501.

Pérez-Urbina, H.; Motik, B.; and Horrocks, I. 2010. Tractable query answering and rewriting under description logic constraints. *J. of Applied Logic* 8(2):186–209.

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

Rodríguez-Muro, M., and Calvanese, D. 2011. Dependencies: Making ontology based data access work in practice. In *Proc. of the 5th Alberto Mendelzon Int. Workshop on Foundations of Data Management (AMW 2011)*, volume 749 of *CEUR Electronic Workshop Proceedings,* http://ceur-ws.org/.

Rosati, R., and Almatelli, A. 2010. Improving query answering over *DL-Lite* ontologies. In *Proc. of the 12th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2010)*, 290–300.

Rosati, R. 2008. Finite model reasoning in *DL-Lite*. In *Proc. of the 5th European Semantic Web Conf. (ESWC 2008)*.

Savo, D. F.; Lembo, D.; Lenzerini, M.; Poggi, A.; Rodríguez-Muro, M.; Romagnoli, V.; Ruzzi, M.; and Stella, G. 2010. MASTRO at work: Experiences on ontology-based data access. In *Proc. of the 23rd Int. Workshop on Description Logic (DL 2010)*, volume 573 of *CEUR Electronic Workshop Proceedings,* http://ceur-ws.org/, 20–31.

Stocker, M., and Smith, M. 2008. Owlgres: A scalable OWL reasoner. In *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008)*, volume 432 of *CEUR Electronic Workshop Proceedings,* http://ceur-ws.org/.