Dependencies: Making Ontology Based Data Access Work in Practice

Mariano Rodríguez-Muro and Diego Calvanese

KRDB Research Centre, Free University of Bozen-Bolzano
Piazza Domenicani 3, Bolzano, Italy
{rodriguez,calvanese}@inf.unibz.it

Abstract. Query answering in Ontology Based Data Access (OBDA) exploits the knowledge of an ontology's TBox to deal with incompleteness of the ABox (or data source). Current query-answering techniques with DL-Lite require exponential size query reformulations, or expensive data pre-processing, and hence may not be suitable for data intensive scenarios. Also, these techniques present severe redundancy issues when dealing with ABoxes that are already (partially) complete. It has been shown that addressing redundancy is not only required for tractable implementations of decision procedures, but may also allow for sizable improvements in execution times. Considering the previous observations, in this paper we present two complementary sets of results that aim at improving query answering performance in OBDA systems. First, we show that we can characterize completeness of an ABox by means of dependencies, and that we can use these to optimize DL-Lite TBoxes. Second, we show that in OBDA systems we can create ABox repositories that appear to be complete w.r.t. a significant portion of any DL-Lite TBox. The combination of these results allows us to design OBDA systems in which redundancy is minimal, the exponential aspect of query answering is notably reduced and that can be implemented efficiently using existing RDBMSs.

1 Introduction

The current approaches to Ontology Based Data Access (OBDA) with lightweight Description Logics (DLs) of the *DL-Lite* family [1] rely on query reformulation. These techniques are based on the idea of using the ontology to rewrite a given query into a new query that, when evaluated over the data sources, returns the certain answers to the original query. Experiments with unions of conjunctive queries (UCQs) have shown that reformulations may be very large, and that the execution of these reformulations suffers from poor performance. This triggered the development of alternative reformulation techniques [6, 9], in which the focus has been on the reduction of the number of generated queries/rules. These techniques have shown some success, however query reformulation in all of them is still worst-case exponential in the size of the original query. Alternative approaches [4] use the expansion of the extensional layer of the ontology (i.e., the ABox) w.r.t. the intensional knowledge (i.e., the TBox) to avoid query reformulation almost entirely. However, the cost of data expansion imposes severe limitations on the system. We believe that approaching the problem of the high cost of query answering in OBDA systems requires a change of focus: namely, from the 'number of queries' perspective, to the perspective that takes into account the 'duplication in the answers' appearing in query results under SQL multiset semantics. Duplication in results is a sign of *redundancy* in the reasoning process; it not only generated not only by the reformulation procedures as traditionally thought, since also techniques based on ABox expansion show this problem. Instead, redundancy is the consequence of ignoring the semantics of the data sources. In particular, when the data in a source (that is used to populate the ABox of the ontology) already satisfies an inclusion assertion of the TBox (i.e., is *complete* w.r.t. such an inclusion assertion), then using that inclusion assertion during query answering might generate redundant answers [8]. It has been noted [3] that it is not hard to find examples in which the runtime of decision procedures can change from exponential to polynomial if redundancy is addressed, and this is also the case in OBDA query answering. In this paper we address both problems, redundancy and the exponential blow-up of query reformulations, by following two complementary directions.

First, we present an approach to take into account completeness of the data with respect to the TBox. We characterize completeness using *ABox dependencies* and show that it is possible to use dependencies to optimize the TBox in order to avoid redundant computations *independently of the reasoning technique*. Second, we focus on how we can optimally complete ABoxes in OBDA systems by relying on the fact that in OBDA systems it is possible to manipulate not only the data, but also the mappings and database schema. This allows us to conceive procedures to store an ABox in a source in such a way that it *appears* to be complete with respect to a significant portion of the TBox, but without actually *expanding* the data. We present two such procedures, one for general and one for 'virtual' OBDA systems, both designed to take advantage of the features of modern RDBMSs effectively. Our results allow for the design of systems that can delegate reasoning tasks (e.g., dealing with hierarchies, existentially quantified individuals, etc.) to stages of the reasoning process where these tasks can be handled most effectively. The result is a (sometimes dramatic) reduction of the exponential runtime and an increase in the quality of the answers due to the reduction of duplication.

The rest of the paper is organized as follows: Section 2 gives technical preliminaries. Section 3 presents our first main contribution, a general technique for optimizing TBoxes w.r.t. dependencies. Section 4 introduces data dependencies in OBDA systems, describing why it is natural to expect completeness of ABoxes. Section 5 presents two techniques for completing ABoxes in OBDA systems, and Section 6 concludes the paper.

2 Preliminaries

In the rest of the paper, 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. In the following, 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.

***DL-Lite* ontologies**. We introduce the DL *DL-Lite*$_{\mathcal{F}}$, on which we base our results. In *DL-Lite*$_{\mathcal{F}}$, 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 assertions of the form $B_1 \sqsubseteq B_2$, called *(positive) inclusions*, $B_1 \sqsubseteq \neg B_2$, called *disjointness assertions*, or $(\mathsf{funct}\ R)$, called *functionality assertions*. An ABox is a finite set of *membership assertions* of the form $A(c)$ or $P(c, c')$, where $c, c' \in \Gamma$.

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 $q(\boldsymbol{x}) \leftarrow \beta(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x}$ is a tuple of distinct variables, called *distinguished*, $\boldsymbol{y}$ is a tuple of distinct variables not occurring in $\boldsymbol{x}$, called *non-distinguished*, and $\beta(\boldsymbol{x}, \boldsymbol{y})$ is a *conjunction* of atoms with variables in $\boldsymbol{x}$ and $\boldsymbol{y}$, whose predicates are atomic concepts and roles of $\mathcal{O}$. We call $q(\boldsymbol{x})$ the *head* of the query and $\beta(\boldsymbol{x}, \boldsymbol{y})$ its *body*. A *union of CQs* (UCQ) is a set of CQs (called disjuncts) with the same head. Given a CQ Q with body $\beta(\boldsymbol{z})$ and a tuple $\boldsymbol{v}$ of constants of the same arity as $\boldsymbol{z}$, we call a *ground instance* of Q the set $\beta[\boldsymbol{z}/\boldsymbol{v}]$ of ground atoms obtained by replacing in $\beta(\boldsymbol{z})$ each variable with the corresponding constant from $\boldsymbol{v}$.

Semantics. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty *interpretation domain* $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$ that assigns to each constant c an element $c^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, to each atomic concept A a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and to each atomic role P a binary relation over $\Delta^{\mathcal{I}}$. Moreover, basic roles and basic concepts are interpreted as follows: $(P^-)^{\mathcal{I}} = \{(o_2, o_1) \mid (o_1, o_2) \in P^{\mathcal{I}}\}$ and $(\exists R)^{\mathcal{I}} = \{o \mid \exists o'.\ (o, o') \in R^{\mathcal{I}}\}$. An interpretation $\mathcal{I}$ is a *model* of $B_1 \sqsubseteq B_2$ if $B_1^{\mathcal{I}} \subseteq B_2^{\mathcal{I}}$, of $B_1 \sqsubseteq \neg B_2$ if $B_1^{\mathcal{I}} \cap B_2^{\mathcal{I}} = \emptyset$, and of $(\mathsf{funct}\ R)$ if for each $o, o_1, o_2 \in \Delta^{\mathcal{I}}$ we have that $(o, o_1) \in R^{\mathcal{I}}$ and $(o, o_2) \in R^{\mathcal{I}}$ implies $o_1 = o_2$. Also, $\mathcal{I}$ is a model of $A(c)$ if $c^{\mathcal{I}} \in A^{\mathcal{I}}$, and of $P(c, c')$ if $(c^{\mathcal{I}}, c'^{\mathcal{I}}) \in P^{\mathcal{I}}$. In *DL-Lite*$_{\mathcal{F}}$, we adopt the Unique Name Assumption (UNA), which enforces that for each pair of constants o_1, o_2, if $o_1 \neq o_2$, then $o_1^{\mathcal{I}} \neq o_2^{\mathcal{I}}$. For a *DL-Lite*$_{\mathcal{F}}$ assertion α (resp., a set Θ of *DL-Lite*$_{\mathcal{F}}$ assertions), $\mathcal{I} \models \alpha$ (resp., $\mathcal{I} \models \Theta$) denotes that $\mathcal{I}$ is a model of α (resp., Θ). A *model of an ontology* $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \models \mathcal{A}$. An ontology is *satisfiable* if it admits a model. An ontology $\mathcal{O}$ *entails* an assertion α, denoted $\mathcal{O} \models \alpha$, if every model of $\mathcal{O}$ is also a model of α. Similarly, for a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$ instead of $\mathcal{O}$. The *saturation* of a TBox $\mathcal{T}$, denoted $sat(\mathcal{T})$, is the set of *DL-Lite*$_{\mathcal{F}}$ assertions α s.t. $\mathcal{T} \models \alpha$. Notice that $sat(\mathcal{T})$ is finite, hence a TBox.

Let $\Gamma_{\mathcal{A}}$ denote the set of constants appearing in an ABox $\mathcal{A}$. The *answer* to a CQ $Q = q(\boldsymbol{x}) \leftarrow \beta(\boldsymbol{x}, \boldsymbol{y})$ 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 $\boldsymbol{c} \in \Gamma_{\mathcal{A}} \times \cdots \times \Gamma_{\mathcal{A}}$ such that there exists a tuple $\boldsymbol{c}' \in \Gamma_{\mathcal{A}} \times \cdots \times \Gamma_{\mathcal{A}}$ such that the ground atoms in $\beta[(\boldsymbol{x}, \boldsymbol{y})/(\boldsymbol{c}, \boldsymbol{c}')]$ are true in $\mathcal{I}$. The answer to an UCQ Q in $\mathcal{I}$ is the union of the answers to each CQ in Q. The *certain answers* to Q in $\mathcal{O}$, denoted $cert(Q, \mathcal{O})$, is the intersection of every $ans(Q, \mathcal{O}, \mathcal{I})$ for all models $\mathcal{I}$ for $\mathcal{O}$. The *answer to Q over an ABox $\mathcal{A}$*, denoted $eval(Q, \mathcal{A})$, is the answers 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 [7]. First we extend interpretations to be able to create object constants from the value constants in a DB $\mathbf{D}$. Given an alphabet Λ of *function symbols* we define the set $\tau(\Lambda, \Gamma_{\mathbf{D}})$ of *object terms* as the set of all terms of the form $\mathbf{f}(d_1, \ldots, d_n)$, where $\mathbf{f} \in \Lambda$, the arity of $\mathbf{f}$ is

n, and $d_1, \ldots, d_n \in \Gamma_{\mathbf{D}}$. We set $\Gamma = \Gamma_{\mathbf{D}} \cup \tau(\Lambda, \Gamma_{\mathbf{D}})$, and we extend the interpretation function so that for each $c \in \tau(\Lambda, \Gamma_{\mathbf{D}})$ we have that $c^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. We extend queries by allowing the use of predicate arguments that are *variable terms*, i.e., expressions of the form $\mathbf{f}(\boldsymbol{t})$, where $\mathbf{f} \in \Lambda$ with arity n and $\boldsymbol{t}$ is an n-tuple of variables or value constants. Given a TBox $\mathcal{T}$ and a DB $\mathbf{D}$, a *mapping (assertion)* m for $\mathcal{T}$ is an expression of the form $\varphi(\boldsymbol{x}) \rightsquigarrow \psi(\boldsymbol{t})$ where $\varphi(\boldsymbol{x})$ is an SQL query over $\mathbf{D}$ with answer variables $\boldsymbol{x}$, and $\psi(\boldsymbol{t})$ is a CQ over $\mathcal{T}$ without non-distinguished variables using variable terms over variables in $\boldsymbol{x}$. We call the mapping *simple* if the body of $\psi(\boldsymbol{t})$ 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 $\psi(\boldsymbol{t})$ 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 $\varphi(\boldsymbol{x}) \rightsquigarrow \psi(\boldsymbol{t})$ w.r.t. a DB $\mathbf{D} = \langle \mathbf{R}, \mathbf{I} \rangle$ if for every tuple $\boldsymbol{v} \in \varphi(\mathbf{I})$ and for every ground atom X in $\psi[\boldsymbol{x}/\boldsymbol{v}]$ we have that: if X has the form $A(\mathbf{f}(\boldsymbol{c}))$, then $(\mathbf{f}(\boldsymbol{c}))^{\mathcal{I}} \in A^{\mathcal{I}}$, and if X has the form $P(\mathbf{f_1}(\boldsymbol{c}_1), \mathbf{f_2}(\boldsymbol{c}_2))$, then $((\mathbf{f_1}(\boldsymbol{c}_1))^{\mathcal{I}}, \mathbf{f_2}(\boldsymbol{c}_2)^{\mathcal{I}}) \in P^{\mathcal{I}}$. 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 [7]). 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 inclusion dependencies only. A *(unary) inclusion dependency* is an assertion of the form $B_1 \sqsubseteq_A B_2$, where B_1 and B_2 are basic concepts. 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: *(i)* if σ is $A_1 \sqsubseteq_A A_2$, then for all $A_1(c) \in \mathcal{A}$ we have $A_2(c) \in \mathcal{A}$; *(ii)* if σ is $\exists R \sqsubseteq_A A$, then for all $R(c, c') \in \mathcal{A}$ we have $A(c) \in \mathcal{A}$; *(iii)* if σ is $A \sqsubseteq_A \exists R$, then for all $A(c) \in \mathcal{A}$ there exists c' such that $R(c, c') \in \mathcal{A}$. *(iv)* 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 $R_2(c, c'') \in \mathcal{A}$. 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 of dependencies Σ *entails* a dependency σ, denoted $\Sigma \models \sigma$, if for every ABox $\mathcal{A}$ s.t. $\mathcal{A} \models \Sigma$ we also have that $\mathcal{A} \models \sigma$. The *saturation* of a set Σ of dependencies, denoted $sat(\Sigma)$, is the set of dependencies σ s.t. $\Sigma \models \sigma$. 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 Optimizing TBoxes w.r.t. Dependencies

In a *DL-Lite*$_{\mathcal{F}}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, the ABox $\mathcal{A}$ may be *incomplete* w.r.t. the TBox $\mathcal{T}$, i.e., there may be assertions $B_1 \sqsubseteq B_2$ in $\mathcal{T}$ s.t. $\mathcal{A} \not\models B_1 \sqsubseteq_A B_2$. When computing the certain answers to queries over $\mathcal{O}$, the TBox $\mathcal{T}$ is used to overcome such incompleteness. However, an ABox may already be (partially) complete w.r.t. $\mathcal{T}$, e.g., an ABox $\mathcal{A}$ satisfying $A_1 \sqsubseteq_A A_2$ is complete w.r.t. $A_1 \sqsubseteq A_2$. While ignoring completeness of an ABox is 'harmless' in the theoretical analysis of reasoning over *DL-Lite*$_\mathcal{F}$ ontologies, in practice, it may introduce *redundancy*, which manifests itself as *containment w.r.t. dependencies* among the disjuncts (CQs) of the perfect reformulation, making the contained disjuncts *redundant*. For example, let $\mathcal{T}$ and $\mathcal{A}$ be as before, and let Q be $q(x) \leftarrow A_2(x)$, then any perfect reformulation of Q must include $q_1 = q(x) \leftarrow A_1(x)$ and $q_2 = q(x) \leftarrow A_2(x)$ as disjuncts. However, since q_1 is contained in q_2 relative to $A_1 \sqsubseteq_A A_2$, we have that q_1 will not contribute new tuples w.r.t. those contributed by q_2.

It is possible to use information about completeness of an ABox, expressed as a set of dependencies, to avoid redundancy in the reasoning process. One place to do this is during query reformulation, using techniques based on 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 reformulation. In particular, given a TBox $\mathcal{T}$ and a set Σ of dependencies, we show how to compute a TBox $\mathcal{T}'$ that is smaller than $\mathcal{T}$ and such that for every query Q the certain answers are preserved if Q is executed over an ABox that satisfies Σ. Specifically, our objective is to determine when an inclusion assertion of $\mathcal{T}$ is *redundant w.r.t.* Σ, and to do so we use the following auxiliary notions.

Definition 1. *Let $\mathcal{T}$ be a TBox, B, C basic concepts, 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 concept 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.*

Intuitively, when there is a $\mathcal{T}$-chain from B to C, the existence of an instance of B in a model 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 2. *Let $\mathcal{T}$ be a TBox, B, C basic concepts, and Σ a set of dependencies. The concept 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 $(B_i \sqsubseteq_A B'_i)_{i=0}^n$. Then, $B \sqsubseteq C$ is* redundant in $\mathcal{T}$ w.r.t. Σ *if (a) it is directly redundant, or (b) there exists $B' \neq B$ s.t. (i) $\mathcal{T} \models B' \sqsubseteq C$, (ii) $B' \sqsubseteq C$ is not redundant in $\mathcal{T}$ w.r.t. Σ, and (iii) $B \sqsubseteq B'$ is directly redundant in $\mathcal{T}$ w.r.t. Σ.*

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 the saturation of $\mathcal{T}$ to obtain a TBox $\mathcal{T}'$ that is equivalent to $\mathcal{T}$ for certain answer computation.

Definition 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 $optim(\mathcal{T}, \Sigma)$, 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 $\mathcal{T}' = optim(\mathcal{T}, \Sigma)$ instead of $\mathcal{T}$ when computing the certain answers to a query follows from the following theorem.

Theorem 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 $cert(Q, \langle \mathcal{T}, \mathcal{A} \rangle) = cert(Q, \langle optim(\mathcal{T}, \Sigma), \mathcal{A} \rangle)$.*

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 [1] for the definition of chase for a *DL-Lite*$_\mathcal{F}$ ontology.) Hence it suffices to show that for every $B \sqsubseteq C$ that is redundant with respect to Σ, $chase(\mathcal{T}, \mathcal{A}) = chase(\mathcal{T} \setminus \{B \sqsubseteq C\}, \mathcal{A})$. We show this by proving that if $B \sqsubseteq C$ is redundant (hence, removed by $optim(\mathcal{T}, \Sigma)$), 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 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 2(*b*), and hence there exists $B' \neq B$ such that $\mathcal{T} \models B \sqsubseteq B'$. 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. □

Complexity and implementation. Due to space limitations, we cannot provide a full description of how to compute $optim(\mathcal{T}, \Sigma)$. We just note that the checks that are required by $optim(\mathcal{T}, \Sigma)$ can be reduced to computing reachability between two nodes in a DAG that represents the reachability relation of the chains in $\mathcal{T}$ and Σ. This operation can be done in linear time.

Consistency checking. Consistency checking may also suffer from redundancy when the ABox is already (partially) complete w.r.t. $\mathcal{T}$. In this case, we need to consider, in addition to inclusion dependencies, also *functional* and *disjointness* dependencies. Due to spaces limitations we cannot provide more details, and just note that using these dependencies it is possible to extend the definitions to generate TBoxes that avoid redundant consistency checking operations.

4 Dependencies in OBDA Systems

The purpose of the current section is to complement our argument w.r.t. completeness of ABoxes by discussing when and why we can expect completeness in OBDA systems. We start by observing that in OBDA systems, ABoxes are constructed, in general, from existing data that resides in some form of data repository. In order to create an ABox, the system requires some form of mappings from the source to the ontology. These may be explicit logical assertions as the ones used in this paper, or they may be implicitly defined through application code. Therefore, the source queries used in these mappings become crucial in determining the structure of the ABox. In particular, any dependencies that hold over the results of these queries will be reflected in the OBDA system as ABox dependencies.

Example 1. Let $\mathbf{R}$ be a DB schema with the relation schema *employee* with attributes *id*, *dept*, and *salary*, that stores information about employees, their salaries, and the department they work for. Let $\mathcal{M}$ be the following mappings:

$$
\begin{array}{ll}
\texttt{SELECT id,dept FROM employee} \rightsquigarrow & \mathit{Employee}(\mathbf{emp}(\texttt{id})) \wedge \\
 & \mathit{WORKS\text{-}FOR}(\mathbf{emp}(\texttt{id}), \mathbf{dept}(\texttt{dept})) \\
\texttt{SELECT id,dept FROM employee} \rightsquigarrow & \mathit{Manager}(\mathbf{emp}(\texttt{id})) \wedge \\
\texttt{WHERE salary > 1000} & \mathit{MANAGES}(\mathbf{emp}(\texttt{id}), \mathbf{dept}(\texttt{dept}))
\end{array}
$$

where *Employee* and *Manager* are atomic concepts and *WORKS-FOR* and *MANAGES* are atomic roles. Then for *every* instance $\mathbf{I}$ of $\mathbf{R}$, the virtual ABox $\mathcal{V} = \langle\langle \mathbf{R}, \mathbf{I}\rangle, \mathcal{M}\rangle$ satisfies the following dependencies:

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

In particular, the dependency in Column 1 follows from the containment relation 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*). ■

Turning our attention to the semantics of the data sources, 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 $\mathcal{T}$, 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 closeness of the semantics of the conceptual model of the DB and the semantics of the TBox, and with the degree in which the DB itself complies to the conceptual model that was used to design it.

Example 2. To illustrate the previous observations we extend Example 1. First we note that the intended meaning of the data stored in $\mathbf{R}$ 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. Then, any TBox that shares some of this semantics will present redundancy. For example, if $\mathcal{T}$ is

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

then the first row of assertions is redundant w.r.t. Σ. Instead, the semantics of the assertions of the second row is not captured by the mappings. In an OBDA system with such components, 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 3. ■

5 Dependency Induction

We focus now on procedures to complete ABoxes with respect to TBoxes. The final objective is to simplify reasoning by diverting certain aspects of the process (e.g., dealing with concept hierarchies and domain and range assertions) from the query reformulation stage to other stages of the query answering process where they can be handled more efficiently. We call these procedures *dependency induction procedures* since their result can be characterized by a set of dependencies that hold in the ABox(es) of the system. Formally, given an OBDA system $\mathcal{O} = \langle \mathcal{T}, \mathcal{V} \rangle$, where $\mathcal{V} = \langle \langle \mathbf{R}, \mathbf{I} \rangle, \mathcal{M} \rangle$, we call a *dependency induction procedure* a procedure that uses $\mathcal{O}$ to compute a virtual ABox $\mathcal{V}'$ such that the number of assertions in $\mathcal{T}$ for which $\mathcal{V}'$ is complete is higher than those for $\mathcal{V}$. An example of a dependency induction procedure is *ABox expansion*, a procedure in which the data in $\mathbf{I}$ is *chased* w.r.t. $\mathcal{T}$. The critical point in dependency induction procedures is the trade-off between the degree of completeness induced, the system's performance, and the cost of the procedure. The purpose of the current section is to present two dependency induction mechanism that provide good trade-offs. Both of them are designed for the case in which the data sources are RDBMSs. Given a TBox $\mathcal{T}$ and a (virtual) ABox $\mathcal{V}$, both techniques are able to generate a system where, if $\mathcal{T} \models B \sqsubseteq A$, then $\mathcal{V}' \models B \sqsubseteq_A A$. Hence, $\mathcal{V}'$ is complete for all *DL-Lite*$_\mathcal{F}$ inferences except those involving mandatory participation assertions, e.g., $B \sqsubseteq \exists R$.

Semantic Index. The first proposal is applicable in the context of general OBDA systems in which we are free to manipulate *any* aspect of the system to improve query answering. The basic idea of the semantic index is to associate a numeric value to each concept of the ontology in such a way that for any two concepts A_1, A_2, if $\mathcal{T} \models A_1 \sqsubseteq A_2$, then the value of A_1 is less than that of A_2. ABox membership assertions are then represented in the DB using these numeric values instead of concept names. This allows one to retrieve most of the implied instances of a concept by posing simple range queries to the DB (which are very efficient in modern RDBMSs). Our proposal is related to a technique for XPath query evaluation known as *Dynamic Intervals* [2], however, while the latter deals with XML trees, we have to deal with concept hierarchies that are DAGs. Formally, a semantic index is defined as follows, where $\mathcal{C}_\mathcal{T}$ denotes the set of atomic concepts of $\mathcal{T}$.

Definition 4. *Given a DL-Lite$_\mathcal{F}$ TBox $\mathcal{T}$,* a semantic index for $\mathcal{T}$ *is a pair of mappings* $\langle idx, range\rangle$ *with* $idx : \mathcal{C}_\mathcal{T} \to \mathbb{N}$ *and* $range : \mathcal{C}_\mathcal{T} \to 2^{\mathbb{N}\times\mathbb{N}}$*, such that for each pair of atomic concepts* A_1*,* A_2 *in* $\mathcal{C}_\mathcal{T}$*, we have that* $\mathcal{T} \models A_1 \sqsubseteq A_2$ *iff there is a pair* $\langle \ell, h\rangle \in range(A_2)$ *such that* $\ell \leq idx(A_1) \leq h$*.*

Using a semantic index $\langle idx, range\rangle$ for a TBox $\mathcal{T}$, we construct $\mathcal{V} = \langle \mathbf{R}, \mathbf{I}\rangle$ with the completeness properties described above, by proceeding as follows. We define a DB schema $\mathbf{R}$ with a universal-like relation $univc[\texttt{c1}, \texttt{idx}]$ for storing ABox concept assertions, and one relation $role_P[\texttt{c1}, \texttt{c2}]$ for each atomic role P, 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 $\langle c, idx(A)\rangle \in univc$ and for each $P(c, c') \in \mathcal{A}$ we have $\langle c, c'\rangle \in role_P$. The schema and the index allow us to define, for each atomic concept A, a set of range queries over $\mathbf{D}$ that retrieves most constants c such that $\mathcal{O} \models A(c)$. E.g., if $range(A) = \{\langle 2, 35\rangle\}$, we define `'SELECT c1 FROM univc WHERE idx >= 2 AND idx <= 35'`. We use these queries to define the mappings of the system as follows[1]: *(i)* for each atomic concept A and each $\langle \ell, h\rangle \in range(A)$, we add the mapping $\sigma_{\ell \leq idx \leq h}(univc) \rightsquigarrow A(c_1)$; *(ii)* for each atomic role P, we add the mapping $role_P \rightsquigarrow P(c_1, c_2)$; *(iii)* for each atomic role P and each atomic concept A such that $\mathcal{T} \models \exists P \sqsubseteq A$ (resp., $\exists P^- \sqsubseteq A$), we add the mapping $\pi_{c_1}(role_P) \rightsquigarrow A(c_1)$ (resp., $\pi_{c_2}(role_P) \rightsquigarrow A(c_2)$).

Note that a semantic index can be trivially constructed by assigning to each concept a unique (arbitrary) value and a set of ranges that covers all the values of subsumed concepts. However, this is not effective for optimizing query answering since the size of $\mathcal{M}$ determines exponentially the size of the final SQL query. To avoid an exponential blow-up, we generate idx and $range$ using the implied concept hierarchy as follows.

Let $\mathcal{T}$ be a TBox, and D the minimal DAG that represents the *implied is-a* relation between all atomic concepts of $\mathcal{T}$ (i.e., the *transitive reduct* of D)[2]. Then we can construct idx in the following way: *(a)* Define a counter $i = 0$; *(b)* Visit the nodes in D in a depth-first fashion starting from the root nodes. At each step and given the node N visited at that step, *(1)* if $idx(N)$ is undefined, set $idx(N) = i$ and $i = i + 1$, *(2)* else if $idx(N)$ is defined, backtrack until the next node for which idx is undefined. Now, to generate $range$ we proceed as follows: *(a)* Visit the nodes in D starting from the leafs and going up. For each node N in the visit, *(1)* if N is a leaf in D, then we set $range(N) = \{\langle idx(N), idx(N)\rangle\}$ *(2)* if N is not a leaf, then we set $range(N) = merge(\{\langle idx(N), idx(N)\rangle\} \cup \bigcup_{N_i \mid N_i \to N \in D} range(N_i))$, where $merge$ is a function that, given a set r of ranges, returns the minimal set r' of ranges that has the same numeric coverage as r, e.g., $merge(\{\langle 5, 7\rangle, \langle 3, 5\rangle, \langle 9, 10\rangle\}) = \{\langle 3, 7\rangle, \langle 9, 10\rangle\}$.

Performance of the Semantic Index. We now describe a preliminary evaluation of the performance of the Semantic Index that was carried out using data from the *Resource Index* (RI) [5]. The RI is an application that offers semantic search services over a collection of 22 well known collections of biomedical data (i.e., documents). The semantics of the search is defined by the hierarchical information of ≈ 200 ontologies, including well known bio-medical ontologies like the Gene Ontology, SNOMEDCT, etc. The basic func-

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

[2] We assume w.l.o.g. that $\mathcal{T}$ does not contain a cyclic chain of basic concept inclusions.

tioning 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*('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 information in the RI's ontologies amounts to ≈ 3 million concepts and ≈ 2.5 million *is-a* assertions. Currently, the number of annotations to be expanded is large, e.g., the annotation process generates ≈ 181 million annotations (≈ 14 GB of data) only for the documents in the *Clinical Trials.gov* (CT) resource collection.

Our experimentation focused on the cost of query answering by *a)* traditional query reformulation, *b)* query answering using a Semantic Index, and *c)* query answering with ABox expansion. To carry out the experimentation we used part of the data from the RI, specifically, we used the full set of *is-a* assertions from the ontology collection and all the document annotations for the CT resource. Also note that since there are no atomic roles in the RI, the Semantic Index technique can generate a complete virtual ABox (i.e., $optim(\mathcal{T}, \Sigma) = \emptyset$). 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. Using this data we created a Semantic Index (time required: 27 s) and created a new table with the CT annotations extended with the Semantic Index. 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) over the CT data. The query returns a total of 2 distinct resources. The results of each technique are as follows[3]: *a)* when rewriting using traditional query reformulation (see [1, 6]), the result is either one SQL query with 467874 disjuncts, or a union of 467874 SQL queries; non of these queries is executable by the engine; *b)* when applying the Semantic Index technique, the result of query reformulation is one 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); *c)* if the ABox is expanded, the TBox is also discarded and the reformulation is equal to the original query; executing this query over the expanded data requires 3 s (0 s if warm). With respect to the cost of the expansion, LePendu et al. indicate that a straightforward expansion for the CT resource collection requires ≈ 7 days, generates an additional ≈ 126 GB of data and, after a careful optimization of the process (including data partitioning, parallelization, etc.) this time can be reduced to ≈ 40 minutes. Given these results, we believe that the Semantic Index is a very promising option for semantic query answering. A system that uses the Semantic Index can provide a better performance than a system that relies only on query reformulation or ABox expansion.

$\mathcal{T}$-Mappings for OBDA. The second technique we present is applicable in the *virtual OBDA* context, where the data should stay in the data sourceand the only aspect of the virtual ABox we can manipulate are the mappings. Our proposal is based on the idea that, given an atomic concept A and a basic concept B, if we want to induce a dependency $B \sqsubseteq_A A$, we have to ensure that the mappings for A retrieve all the data retrieved by those for B. We call the technique $\mathcal{T}$-mappings, and define it formally as follows.

[3] More information about these tests, including the SQL queries can be found at `https://babbage.inf.unibz.it/trac/obdapublic/wiki/resourceindex`

Definition 5. *Given a TBox $\mathcal{T}$ and a virtual ABox $\mathcal{V} = \langle \mathbf{D}, \mathcal{M} \rangle$, a $\mathcal{T}$-mapping for $\mathcal{T}$ w.r.t. $\mathcal{V}$ is a virtual ABox $\mathcal{V}' = \langle \mathbf{D}, \mathcal{M}' \rangle$ such that for every ABox assertion α such that $\mathcal{V} \models \alpha$, we have that $\mathcal{V}' \models \alpha$, and for each ground ABox assertion α such that $\langle \mathcal{T}, \mathcal{V} \rangle \models \alpha$ we have that $\mathcal{V}' \models \alpha$.*

A $\mathcal{T}$-mapping $\mathcal{M}'$ can be constructed trivially by using existing mappings to generate new mappings that take into account the implications of $\mathcal{T}$ as follows: Initialize $\mathcal{M}' = \emptyset$, and for each atomic concept A, *(i)* for each mapping $\varphi \rightsquigarrow A'(\mathbf{f}(\boldsymbol{x})) \in \mathcal{M}$ such that $\mathcal{T} \models A' \sqsubseteq A$, add $\varphi \rightsquigarrow A(\mathbf{f}(\boldsymbol{x}))$ to $\mathcal{M}'$, and *(ii)* for each mapping $\varphi \rightsquigarrow P(\mathbf{f}(\boldsymbol{x}), \mathbf{g}(\boldsymbol{y})) \in \mathcal{M}$ such that $\mathcal{T} \models \exists P \sqsubseteq A$ (resp., $\mathcal{T} \models \exists P^- \sqsubseteq A$), add $\varphi \rightsquigarrow A(\mathbf{f}(\boldsymbol{x}))$ (resp., $\varphi \rightsquigarrow A(\mathbf{g}(\boldsymbol{y}))\}$) to $\mathcal{M}'$. However, such $\mathcal{T}$-mappings will not be efficient since the increased size of $\mathcal{M}$ will trigger a worst-case exponential blow-up during SQL query generation.

To avoid this situation we aim at creating succinct mappings by resorting to SQL query analysis as follows. Let $\mathbf{D} = \langle \mathbf{R}, \mathbf{I} \rangle$ be a DB instance, $\Sigma_\mathbf{D}$ a set of dependencies satisfied by $\mathbf{D}$, $SQ = \{\varphi_1, \ldots, \varphi_n\}$ a set of SQL queries defined over $\mathbf{R}$, and $\boldsymbol{x}$ a list of attribute names, then $mergesql(SQ, \Sigma_\mathbf{D}, \boldsymbol{x})$ is the function that returns a new SQL query φ that is *equivalent* under $\Sigma_\mathbf{D}$ to the union of all $\varphi_i \in SQ$ projected on $\boldsymbol{x}$. E.g., if $SQ = \{$ `'SELECT id FROM employee'`, `'SELECT id FROM employee JOIN information ON id'`, `'SELECT id FROM external'`$\}$ and $\Sigma_\mathbf{D}$ is the inclusion dependency *external*$[id] \subseteq$ *employee*$[id]$, then $mergesql(SQ, \Sigma_\mathbf{D}, [id]) =$ `'SELECT id FROM employee'`. Using *mergesql*, we define two more auxiliary functions, *mergemc* and *mergemr*, which allow us to *merge* sets of mappings. Let S_C be a set of mappings for atomic concepts, S_R a set of mappings for atomic roles, and A an atomic concept. Let F be the set of all unique (up to variable renaming) variable terms $\mathbf{f}(\boldsymbol{x})$ that are used in a mapping in S_C, then $mergemc(S_C, \Sigma_\mathbf{D}, A) = \bigcup_{\mathbf{f}_i(\boldsymbol{x}_i) \in F} \{mergesql(SQ_{\mathbf{f}_i}, \Sigma_\mathbf{D}, \boldsymbol{x}_i) \rightsquigarrow A(\mathbf{f}_i(\boldsymbol{x}_i))\}$, where $SQ_{\mathbf{f}_i}$ is the set of SQL queries φ of all the mappings $\varphi \rightsquigarrow A_1(\mathbf{f}_i(\boldsymbol{x})) \in S_C$ where $\boldsymbol{x}_i$ and $\boldsymbol{x}$ are unifiable. Let F_1 (resp., F_2) be the set of all unique (up to variable renaming) variable terms $\mathbf{f}(\boldsymbol{x})$ that are used as first (resp., second) component in the head of a mapping in S_R, then $mergemr(S_R, \Sigma_\mathbf{D}, 1, A) = \bigcup_{\mathbf{f}_i \in F_1} \{mergesql(SQ_{\mathbf{f}_i}, \Sigma_\mathbf{D}, \boldsymbol{x}_i) \rightsquigarrow A(\mathbf{f}_i(\boldsymbol{x}_i))\}$ (resp., $mergemr(S_R, \Sigma_\mathbf{D}, 2, A) = \bigcup_{\mathbf{f}_i \in F_2} \{mergesql(SQ_{\mathbf{f}_i}, \Sigma_\mathbf{D}, \boldsymbol{x}_i) \rightsquigarrow A(\mathbf{f}_i(\boldsymbol{x}_i))\}$), where $SQ_{\mathbf{f}_i}$ is the set of SQL queries φ of all the mappings $\varphi \rightsquigarrow P(\mathbf{f}_i(\boldsymbol{x}), \mathbf{g}(\boldsymbol{y}))$ (resp., $\varphi \rightsquigarrow P(\mathbf{g}(\boldsymbol{y}), \mathbf{f}_i(\boldsymbol{x}))$) in S_R where $\boldsymbol{x}_i$ and $\boldsymbol{x}$ unify. Let $M(A)$ (resp., $M(P)$) be the set of all mappings for the atomic concept A (resp., for the atomic role P) in $\mathcal{M}$. Then, given a TBox $\mathcal{T}$ and a set $\mathcal{M}$ of mappings, the $\mathcal{T}$-mappings for $\mathcal{M}$ w.r.t. $\mathcal{T}$ is defined as $\mathcal{M}' = \bigcup_{P \in V} M(P) \cup \bigcup_{A \in V} mergemc(m_A, \Sigma_\mathbf{D}, A)$, with $m_A = \bigcup_{\mathcal{T} \models A_i \sqsubseteq A} M(A_i) \cup mergemr(mr, \Sigma_\mathbf{D}, 1, A) \cup mergemr(mr^-, \Sigma_\mathbf{D}, 2, A)$, $mr = \bigcup_{\mathcal{T} \models \exists P_i \sqsubseteq A} M(P_i)$, and $mr^- = \bigcup_{\mathcal{T} \models \exists P_i^- \sqsubseteq A} M(P_i)$. Intuitively, this is the original role mappings in $\mathcal{M}$ union the merging of all the relevant mappings for each A (w.r.t. TBox implication).

Example 3. Let $\mathbf{D}$ be the DB in Example 1, $\mathcal{M}$ the simple version of the mappings in that example (i.e., 4 simple mappings), and $\mathcal{T}$ the TBox in Example 2. Then the $\mathcal{T}$-mappings for $\mathcal{T}$ is $\mathcal{M}'$ consisting of all the mappings in $\mathcal{M}$ plus the additional mapping $m =$ `'SELECT id,dept FROM employee'` $\rightsquigarrow$ *Department*$(\mathbf{dept}(\texttt{dept}))$ which we now explain. Note that $\exists$*WORKS-FOR*$^-$ and $\exists$*MANAGES*$^-$ are the only subsumees of *Department* w.r.t. $\mathcal{T}$ and hence, only their mappings (let them be m_1, m_2) are considered for the mappings of *Department* by $mergemc$. Then given the containment between the queries of m_1 and m_2, the output of $mergemc(\{m_1, m_2\}, \emptyset, \textit{Department})$ is m.

6 Conclusions and Future Work

In this paper we focused on issues of redundancy and performance in OBDA systems. Several directions can be taken starting from the ideas presented here. First, although TBox pre-processing is the best place to first address redundancy, it is also necessary to apply redundancy elimination during reasoning, e.g., during query rewriting. Second, redundancy may also appear during consistency checking, i.e., when an ABox is sound w.r.t. to the TBox; this situation can also be characterized by ABox dependencies and should be addressed. With respect to the evaluation of our proposals, further experimentation is still required. In particular, it is necessary to provide a comprehensive benchmark of the techniques discussed in this paper in comparison to other proposals. As future work we are now exploring extensions of the techniques to more expressive languages, e.g., *DL-Lite*$_{\mathcal{A}}$ which supports also role inclusions, and SPARQL queries over RDFS ontologies. In this last context, we believe that our techniques can be used to provide high-performance SPARQL end points with sound and complete RDFS entailment regime support, without relying on inference materialization, as is usually done.

Acknowledgements. We thank Dr. Paea LePendu for providing us with the RI data and supporting us in understanding this application. This research has been partially supported by the EU under the ICT Collaborative Project ACSI grant n. FP7-257593.

References

1. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. of Automated Reasoning*, 39(3):385–429, 2007.
2. D. DeHaan, D. Toman, M. P. Consens, and M. T. Özsu. A comprehensive XQuery to SQL translation using dynamic interval encoding. In *Proc. of ACM SIGMOD*, pages 623–634, 2003.
3. G. Gottlob and C. Fermüller. Removing redundancy from a clause. *Artif. Intell.*, 61(2), 1993.
4. R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The combined approach to query answering in *DL-Lite*. In *Proc. of KR 2010*, 2010.
5. P. LePendu, N. Noy, C. Jonquet, P. Alexander, N. Shah, and M. Musen. Optimize first, buy later: Analyzing metrics to ramp-up very large knowledge bases. In *Proc. of ISWC 2010*, volume 6496 of *LNCS*, pages 486–501. Springer, 2010.
6. H. Pérez-Urbina, B. Motik, and I. Horrocks. Tractable query answering and rewriting under description logic constraints. *J. of Applied Logic*, 8(2):186–209, 2010.
7. A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. on Data Semantics*, X:133–173, 2008.
8. M. Rodríguez-Muro. *Tools and Techniques for Ontology Based Data Access in Lightweight Description Logics*. PhD thesis, KRDB Research Centre for Knowledge and Data, Free Univ. of Bozen-Bolzano, 2010.
9. R. Rosati and A. Almatelli. Improving query answering over *DL-Lite* ontologies. In *Proc. of KR 2010*, 2010.