kyrie2: Query Rewriting under Extensional Constraints in $\mathcal{ELHIO}$

Jose Mora[1,2], Riccardo Rosati[1], and Oscar Corcho[2]

[1] Dipartimento di Ingegneria Informatica, Automatica e Gestionale Antonio Ruberti Sapienza Università di Roma, Italy
lastname@dis.uniroma1.it

[2] Ontology Engineering Group, Departamento de Inteligencia Artificial, Facultad de Informática,Universidad Politécnica de Madrid, Spain
{jmora,ocorcho}@fi.upm.es

Abstract. In this paper we study query answering and rewriting in ontology-based data access. Specifically, we present an algorithm for computing a perfect rewriting of unions of conjunctive queries posed over ontologies expressed in the description logic $\mathcal{ELHIO}$, which covers the OWL 2 QL and OWL 2 EL profiles. The novelty of our algorithm is the use of a set of ABox dependencies, which are compiled into a so-called EBox, to limit the expansion of the rewriting. So far, EBoxes have only been used in query rewriting in the case of DL-Lite, which is less expressive than $\mathcal{ELHIO}$. We have extensively evaluated our new query rewriting technique, and in this paper we discuss the tradeoff between the reduction of the size of the rewriting and the computational cost of our approach.

Keywords: Ontology-Based Data Access, Query Rewriting, Reasoning, EBox.

1 Introduction

In Ontology Based Data Access (OBDA) [1], *ontologies* are used to superimpose a conceptual layer as a view to an underlying *data source*, which is usually a relational database. The conceptual layer consists of a TBox, i.e. a set of axioms expressed in a Description Logic (DL). This layer abstracts away from how that information is maintained in the data layer and may provide inference capabilities. The conceptual layer and the data source layer are connected through *mappings* that specify the semantic relationship between the database schema terms and the terms in the TBox.

Query rewriting is currently the most important reasoning technique for OBDA. It consists in transforming a query posed in ontological terms into another query expressed over the underlying database schema. The rewritten query allows for obtaining the *certain answers* to the original query, i.e. results explicitly stated for the query in the database and those that are entailed by the TBox. To do so, the rewritten query "encodes" the intensional knowledge expressed by the TBox and the mappings [1].

Recently, some approaches [2,3,4] have proposed the use of *ABox dependencies*, or *extensional constraints*, to optimise query rewriting in OBDA. An extensional constraint is an axiom (in the TBox language) that the data are known to satisfy. As such, it can be viewed as an integrity constraint for the OBDA system. Such constraints can be automatically derived from OBDA specifications, in particular, they can be deduced from the mappings and from the integrity constraints in the source database [4]. Following [3], we call *EBox* a set of extensional constraints. For ontologies expressed in the logic DL-Lite$_A$, EBoxes can be used to optimise reasoning and query rewriting in OBDA [2,3]. In fact, since extensional constraints express forms of completeness of the data, they can be used during query rewriting in a complementary way with respect to the usual TBox axioms, allowing for significant simplifications of the rewritten query.

In this paper we explore the application of an EBox to the rewriting process performed by kyrie [5], which deals with the expressive DL $\mathcal{ELHIO}$ by performing resolution in several stages with some optimisations.

The contributions of the paper are the following:

1. *Extension of the kyrie algorithm.* We define a new query rewriting algorithm for $\mathcal{ELHIO}$. The algorithm is based on kyrie, and takes into account, besides the TBox, the presence of an $\mathcal{ELHIO}$ EBox. This extension is inspired by Prexto [3], a query rewriting algorithm for DL-Lite$_R$: however, such an extension is technically challenging, due to the expressiveness of $\mathcal{ELHIO}$.
2. *Extension of a query rewriting benchmark.* We extend an existing benchmark for the evaluation of query rewriting in OBDA [6], considering EBoxes in addition to TBoxes, so as to experimentally evaluate the use of EBoxes in $\mathcal{ELHIO}$ ontologies.
3. *Implementation and experimental evaluation.* We perform an experimental analysis of the new query rewriting algorithm. Our results show the effectiveness of using EBoxes in the optimisation of query rewriting, and highlight some interesting properties of the similarity between TBox and EBox.

This paper is structured as follows. In Section 2 we briefly recall the DL $\mathcal{ELHIO}$ and extensional constraints and the state of the art is briefly summarized. In Section 3 we present the kyrie2 query rewriting algorithm for $\mathcal{ELHIO}$. The operations performed by the algorithm are formalised as a set of propositions in Section 4. Finally, Section 5 and Section 6 contain, respectively, the evaluation of our proposal and some conclusions drawn from it[1].

2 Preliminaries

We briefly recall Horn clauses, $\mathcal{ELHIO}$, OBDA systems and extensional constraints.

Horn Clauses. Following [7,5], our technique makes use of a representation of DL axioms as Horn clauses. A Horn clause (or Horn rule) is an expression of the form $\beta_0 \leftarrow \beta_1, \ldots, \beta_n$, with $n \geq 0$ and where each term appearing in each of the atoms $\beta_0, \beta_1, \ldots, \beta_n$ may be a constant c from an alphabet of constant symbols, a variable x from an alphabet of variable symbols, or a unary Skolem function $f(x)$ (where f is from an alphabet of function symbols) whose argument is a variable. Clauses are safe, i.e., all variables occurring in β_0 (which is called the clause head) also occur in $\beta_1, \ldots, \beta_n$ (which is called the clause body). The arity of the clause is the number of arguments of

[1] Due to space constraints, proofs of theorems are available at `http://j.mp/kyrieproofebox`

its head atom. A clause is Boolean if it has no variables in its head. We say that an atom β_a subsumes another atom β_b ($\beta_a \succeq_s \beta_b$) if there is some unification of its variables μ such that $\mu\beta_a = \beta_b$. A Horn clause subsumes another Horn clause if after some variable renaming both of their heads are equal and there is some unification such that all the atoms in the body of the subsuming clause unify with some atom in the body of the subsumed clause, i.e. $\forall \gamma_a, \gamma_b.(head(\gamma_a) = head(\gamma_b) \wedge \exists \mu.\forall \beta_i \in body(\gamma_a).\exists \beta_j \in body(\gamma_b).\mu\beta_i = \beta_j) \rightarrow \gamma_a \succeq_s \gamma_b$.

Let R be the Horn clause $\beta_0 \leftarrow \beta_1, \ldots, \beta_n$ and let x be the variables occurring in R. We define $FO(R)$ as the first-order sentence $\forall x(\beta_0 \vee \neg\beta_1 \vee \ldots \vee \neg\beta_n)$. Moreover, given a set of Horn clauses Σ, we define $FO(\Sigma)$ as $\bigcup_{R \in \Sigma} FO(R)$.

OBDA Systems. An OBDA system [1] allows for accessing a set of data sources $\mathcal{D}$ using an ontology composed of TBox and ABox $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ as a view for the data in $\mathcal{D}$. To do this, a set of mappings $\mathcal{M}$ is normally used to map the information in $\mathcal{D}$ to the elements in the TBox $\mathcal{T}$ [8].

In $\mathcal{ELHIO}$, concept (C) and role (R) expressions are formed according to the following syntax (where A denotes a concept name, P denotes a role name, and a denotes an individual name):

$$\begin{aligned} C &::= A \mid C_1 \sqcap C_2 \mid \exists R.C \mid \{a\} \\ R &::= P \mid P^- \end{aligned}$$

An $\mathcal{ELHIO}$ axiom is an expression of the form $C_1 \sqsubseteq C_2$, $C \sqsubseteq \bot$, $R_1 \sqsubseteq R_2$ or $R \sqsubseteq \bot$ where C_1, C_2 are concept expressions and R_1, R_2 are role expressions (as usual, $\bot$ denotes the empty concept). An $\mathcal{ELHIO}$ TBox $\mathcal{T}$ is a set of $\mathcal{ELHIO}$ axioms.

An OBDA system is a pair $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is an $\mathcal{ELHIO}$ TBox, and $\mathcal{A}$ is an ABox, i.e., a set of ground atoms, representing the pair $\langle \mathcal{M}, \mathcal{A} \rangle$.

Notably, each $\mathcal{ELHIO}$ axiom corresponds to a set of Horn clauses [7,5]. We can thus define the semantics of OBDA systems by using the clause translation of an $\mathcal{ELHIO}$ TBox into a set of clauses. More precisely, given an $\mathcal{ELHIO}$ axiom ψ, we denote by $\tau_c(\psi)$ the set of clauses corresponding to ψ, and given a TBox $\mathcal{T}$, we denote by $\tau_c(\mathcal{T})$ the set of clauses corresponding to $\mathcal{T}$. Then, the set of models of an OBDA system $\langle \mathcal{T}, \mathcal{A} \rangle$ is the set of models of the first-order theory $FO(\tau_c(\mathcal{T})) \cup \mathcal{A}$.

We refer the reader to [5] for more details on the translation of $\mathcal{ELHIO}$ axioms into Horn clauses. From now on, we assume that the OBDA system $\langle \mathcal{T}, \mathcal{A} \rangle$ is consistent, i.e., has at least one model. The extension of our results to possibly inconsistent OBDA systems is trivial.

The OBDA system allows for accessing the data by posing queries and returning the *certain answers* to these queries. As usual in OBDA, we consider unions of conjunctive queries. A conjunctive query (CQ) is a function-free Horn clause. As usual, we assume that the head predicate q of the CQ does not occur in $\langle \mathcal{T}, \mathcal{A} \rangle$. For ease of exposition, we assume that constants may not occur in the head of a CQ. A union of conjunctive queries (UCQ) is a finite, non-empty set of CQs having the same head predicate q and the same arity.

The set of *certain answers* for a UCQ q posed to a system $\langle \mathcal{T}, \mathcal{A} \rangle$, denoted by $\Phi^q_{\langle \mathcal{T}, \mathcal{A} \rangle}$, is the set of tuples of constants $t_1, \ldots, t_n$ such that the atom $q(t_1, \ldots, t_n)$ holds in all models of $FO(\Sigma \cup q) \cup \mathcal{A}$, where $\Sigma = \tau_c(\mathcal{T})$.

Query Rewriting in OBDA. The main approaches to query answering in OBDA are based on query rewriting techniques whose goal is to compute so-called perfect rewritings. A *perfect rewriting* for a UCQ q and a TBox $\mathcal{T}$ is a query q' such that, for every ABox $\mathcal{A}$, $\Phi^{q}_{\langle \mathcal{T}, \mathcal{A} \rangle} = \Phi^{q'}_{\langle \emptyset, \mathcal{A} \rangle}$ (i.e., the TBox is "encoded" by the rewritten query).

In OBDA, special attention has been paid in these systems to ontology languages that are *first-order (FO) rewritable* [9,10], i.e. such that UCQs always admit a perfect rewriting expressed in first-order logic. $\mathcal{ELHIO}$ falls out of this expressiveness, since recursive Datalog is sometimes needed to express perfect rewritings of UCQs, i.e., it is *Datalog-rewritable*.

Related Systems. Several systems have been implemented to perform query rewriting, these systems and their main characteristics including the expressiveness for the aforementioned Σ are summarised in Table 1. A more detailed description of these systems and the logics they handle can be found in [6]. The interest in this kind of systems is shown by commercial applications like Stardog[2], which can perform query answering with several reasoning levels (RDFS, QL, RL, $\mathcal{EL}$ and DL).

Table 1. Main systems for OBDA query rewriting in the state of the art

System	Input	Output	Year	Reference
Quonto	DL-Lite$_R$	UCQ	2007	Calvanese et al. [9]
REQUIEM	$\mathcal{ELHIO}^{\neg}$	Datalog or UCQ	2009	Pérez-Urbina et al. [7]
Presto	DL-Lite$_R$	Datalog	2010	Rosati and Almatelli [11]
Rapid	DL-Lite$_R$[3]	Datalog or UCQ	2011	Chortaras et al. [12]
Nyaya	*Datalog*$^{\pm}$	UCQ	2011	Gottlob et al. [10]
Venetis'	DL-Lite$_R$	UCQ	2013	Venetis et al. [13]
Prexto	DL-Lite$_R$and EBox	Datalog or UCQ	2012	Rosati [3]
Clipper	Horn-$\mathcal{SHIQ}$	Datalog	2012	Eiter et al. [14]
kyrie	$\mathcal{ELHIO}^{\neg}$	Datalog or UCQ	2013	Mora and Corcho [5]

EBoxes and Query Rewriting. Description logic (DL) ontologies are usually decomposed into ABox (assertional box) and TBox (terminological box). The former includes the assertions or facts, corresponding to the individuals, constants or values (i.e. the extension) of some predicates. The latter is a set of DL axioms that describe the concepts and predicates in the ontology and how they are related. These DL axioms can be converted to rules or implications (or data dependencies) in first order logic (more expressive) and to some extent in Datalog (adding Skolem functions when needed).

Extensional constraints, also known as *ABox dependencies*, are assertions that restrict the syntactic form of allowed or admissible ABoxes in an OBDA system. These assertions have the form of the usual TBox axioms and are interpreted as integrity constraints over the ABox, i.e. under a closed-world assumption instead of the usual open-world assumption of DLs. For example, for an ABox $\mathcal{A}$ that satisfies some EBox $\mathcal{E}$ and expressions C_1, C_2 in $\mathcal{A}$; if $\mathcal{E} \models C_1 \sqsubseteq C_2$ then $\{x_1 \mid C_1(x_1) \in \mathcal{A}\} \subseteq \{x_2 \mid C_2(x_2) \in \mathcal{A}\}$. A set of such assertions is called an extensional constraint box (EBox) [4].

[2] `http://docs.stardog.com/owl2/`

[3] Close to OWL2 QL, $B_1 \sqsubseteq \exists R.B_2$ axioms are supported.

As shown in [3], extensional constraints can be used to simplify the perfect rewriting of a query, because such constraints may imply that some parts of the query are actually redundant. We can see this more easily with an example. For example we may have the following TBox:

$$\begin{array}{rlrl} UndergradStudent & \sqsubseteq Student & MasterStudent & \sqsubseteq GradStudent \\ PhDStudent & \sqsubseteq GradStudent & IndustryMasterStudent & \sqsubseteq MasterStudent \\ GradStudent & \sqsubseteq Student & ResearchMasterStudent & \sqsubseteq MasterStudent \\ BachelorStudent & \sqsubseteq UndergradStudent & & \end{array}$$

And the following EBox:

$$\begin{array}{rlrl} IndustryMasterStudent & \sqsubseteq GradStudent & Student & \sqsubseteq \bot \\ ResearchMasterStudent & \sqsubseteq GradStudent & BachelorStudent & \sqsubseteq \bot \\ PhDStudent & \sqsubseteq GradStudent & MasterStudent & \sqsubseteq \bot \end{array}$$

And we may want to retrieve a list of all the students.

We can consider an ABox that satisfies the previous EBox, for example an ABox with the following individuals:

- $UndergradStudent$: Al
- $GradStudent$: Ben, Don, Ed
- $ResearchMasterStudent$: Ben
- $IndustryMasterStudent$: Cal
- $PhdStudent$: Don

Querying for the most general concept ($Student$) would yield no results. Querying for the most specific concepts ($BachelorStudent$, $ResearchMasterStudent$, $IndustryMasterStudent$ and $PhdStudent$) requires four queries and yields an incomplete answer, missing Ed and Al in the example. Finally querying for all concepts would provide all answers, but that implies eight queries (one for each concept) and retrieving some duplicates. In this case the duplicates are Ben and Don. Duplicated answers have no impact on the correctness of the answer set, but they are a big burden in the efficiency of the process when considering more complex queries and ontologies. In particular, in the example we only need three queries (as opposed to eight) to retrieve all answers, querying respectively for instances of $UndergradStudent$, $GradStudent$ and $IndustryMasterStudent$, since the EBox states that the ABox extension of every other concept is either empty or contained in $GradStudent$. There are therefore six queries that are only a waste of computational resources in the query answering.

A naïve algorithm could generate the perfect rewriting and then reduce it by checking for subsumption with the EBox. However, such a naïve algorithm could have a prohibitive cost for large rewritings and would only be applicable over non-recursive rewritings. In the following sections we will show that it is possible to face more complex scenarios and handle them better than with such a naïve algorithm.

This example illustrates that the combination of ABoxes that are already (partially) complete and a complete query rewriting on the TBox causes redundancy in the results, which is a burden for efficiency. Hence, the characterization of ABox completeness as a set of dependencies can serve to optimise TBoxes, and create ABox repositories that appear to be complete [2]. Additional optimisations can be done with the Datalog query before unfolding it into a UCQ, and finally with the UCQ, reducing redundancy at every step. For instance, in our example we have in the EBox that $PhDStudent \sqsubseteq GradStudent$ just like in the TBox. Therefore, we do not need to consider this axiom in the TBox when retrieving students: the ABox is complete in that sense and no $GradStudent$ needs to be obtained from $PhDStudent$.

Using the EBox, the perfect rewriting can be reduced along with the inference required for its generation. We can redefine the perfect rewriting in the presence of EBoxes as follows [3]: a perfect rewriting for a UCQ q and a TBox $\mathcal{T}$ under an EBox $\mathcal{E}$ is a query q' such that, for every ABox $\mathcal{A}$ that satisfies $\mathcal{E}$, $\Phi^{q}_{\langle \mathcal{T},\mathcal{A}\rangle} = \Phi^{q'}_{\langle \emptyset,\mathcal{A}\rangle}$.

3 Using Extensional Constraints in Query Rewriting

In this section we present the kyrie2 algorithm, providing an overview of the previous kyrie algorithm and detailing the use of extensional constraints. Extensional constraints can be used both in the preprocessing stage, performed before queries are posed to the system, and in the main algorithm for the rewriting of queries when they are available. We conclude this section with the algorithm that prunes a Datalog program (or a UCQ as a specific case of Datalog program) using the available extensional constraints.

Overview of the Technique. kyrie2 extends the earlier kyrie algorithm [5] to handle EBoxes. The original kyrie algorithm obtains a set of clauses Σ from the TBox $\mathcal{T}$ and a query q and performs resolution on this set of clauses.

The usual operations performed in these algorithms are equal to those in kyrie. Here we briefly describe them, a more detailed description can be found in [5]:

- *Saturate* performs a saturation of a set of clauses using a selection function to guide the atoms that should be unified in the resolution. The selection function may be:
  - *sfRQR* is the selection function used in REQUIEM, it avoids the unification of unary predicates with no function symbols to produce a Datalog program.
  - *sfAux* selects auxiliary predicates to perform the inferences in which these predicates may participate and remove them if possible.
  - *sfSel(p)* selects the predicate p.
  - *sfNoRec(P)* selects all predicates except those that included in the set of predicates P (used to avoid infinite resolution on recursive predicates).

  Additionally, the saturation algorithm has a parameter (p, s or u):
  - p preserves the clauses that have been used in the resolution, contrarily other modes do not preserve these clauses, e.g. clauses with functional symbols are removed to obtain a Datalog program.
  - s separates the clauses that are obtained anew from the old ones, returning only those that are new, e.g. when saturating the query with the clauses derived from the TBox all produced clauses will be query clauses.
  - u for unfolding, this method does not skip the cleaning stage and does not separate the results.
- *Condensate* is used to condensate clauses, i.e. remove redundant atoms.
- *RemoveSubsumed* removes clauses that are subsumed in a set of clauses.

There are three main stages in which resolution is performed:

- *Preprocessing* is performed once for the $\mathcal{ELHIO}$ TBox ($\mathcal{T}$), before any query is posed to the system. In this stage some inferences are materialised to save time later and the set of clauses Σ is generated according to the TBox.

- *Saturation* is performed when the query arrives: the query is added to Σ, then functional symbols are removed from Σ, reducing Σ to a Datalog program (i.e., a function-free set of Horn clauses).
- *Unfolding* is performed partially or completely, depending on the respective presence or absence of recursive predicates in the Datalog rewriting.

In kyrie2, we add a further operation in each of these stages, highlighted in the corresponding algorithms. In this paper, we focus of this new and additional operation, referring the reader to [5] for details on the other aspects of the algorithm. The new operation makes use of the EBox to infer extensional subsumption between atoms and between clauses. Atom subsumption in a conjunction of atoms means that the values for one are a subset of the values for another (the most general atom is eliminated from the conjunction). Clause subsumption in a disjunction of clauses means that the values provided by a clause for a predicate are a subset of the values provided by some other clause (the most specific clause is eliminated from the disjunction). In other words, the new operation detects extensional redundancy in the set of clauses, thus allowing for reducing the size of the initial set of clauses, the subsequent Datalog program, and the final UCQ. Since, for technical reasons, the EBox is represented in two different ways in the algorithm (both as a set of standard DL axioms and as a set of clauses), this operation is defined and executed on both representations.

Before introducing the algorithms, we give two preliminary and analogous definitions of graphs relative to an $\mathcal{ELHIO}$ TBox and to a set of Horn clauses, respectively.

Definition 1. *We define $dlgraph(\mathcal{T})$, the* axiom graph *for an $\mathcal{ELHIO}$ TBox $\mathcal{T}$, as the directed graph (V, W) such that: (i) for each axiom $\psi \in \mathcal{T}$, $\psi \in V$; and (ii) for each pair (ψ_a, ψ_b) such that $\psi_a, \psi_b \in \mathcal{T}$ if there is a predicate p such that $p \in RHS(\psi_b)$ and $p \in LHS(\psi_a)$ then $(\psi_a, \psi_b) \in W$. Where LHS and RHS are respectively the left and the right hand sides of the axiom.*

Definition 2. *We define $cgraph(\Sigma)$, the* clause graph *for a set of clauses Σ, as the directed graph (V, W) such that: (i) for each clause $\gamma \in \Sigma$ then $\gamma \in V$; and (ii) for each pair (γ_a, γ_b) such that $\gamma_a, \gamma_b \in \Sigma$ and $\exists p.p \in body(\gamma_b) \wedge p \in head(\gamma_a)$ then $(\gamma_a, \gamma_b) \in W$.*

Both graph notions are equivalent for our purposes, the syntactical differences are due to the stages in the algorithm where each of these notions will be used.

Our algorithms will use both the DL and the clause representation of TBoxes and EBoxes (obtained from the DL syntax through the function τ_c). Therefore, from now on we will use the terms TBox, EBox and OBDA system for both kinds of representations, and will use the symbols Σ, E and Γ to denote, respectively, a TBox, an EBox, and an OBDA system in the Horn clause representation.

Preprocessing. Algorithm 1 constitutes a preprocessing stage on the TBox and the EBox before any query is available. The algorithm removes, through the function `delEBoxSCC`, the strongly connected components (SCCs) of the TBox graph that are implied by the EBox and do not receive incoming connections, i.e. for all axioms ψ_b in the SCC there is no ψ_a in the TBox such that (ψ_a, ψ_b) is in the set of edges of

Algorithm 1. kyrie2 preprocess algorithm

```
Input: $\mathcal{ELHIO}$ TBox $\mathcal{T}$, $\mathcal{ELHIO}$ EBox $\mathcal{E}$
Output: TBox $\Sigma$, EBox $E$, minimal sets of recursive predicates $R_\Sigma$ and $R_E$
1 $T' = \texttt{delEBoxSCC}(\mathcal{T}, \mathcal{E})$
2 $\Sigma = \tau_c(T')$
3 $E = saturate(\texttt{p}, sfNonRec(R_E), \tau_c(\mathcal{E}), \emptyset)$
4 $\langle R_\Sigma, R_E \rangle = \texttt{reducedRecursiveSets}(\Sigma, E)$
5 $\Sigma = saturate(\texttt{p}, sfRQR, \Sigma, \emptyset)$
6 $\Sigma = saturate(\texttt{s}, sfAux, \Sigma, \emptyset)$
7 $\Sigma = removeSubsumed(condensate(\Sigma))$
8 return $\langle \Sigma, E, R_\Sigma, R_E \rangle$
```

Algorithm 2. Remove extensionally implied strongly connected components (SCCs) of axioms: `delEBoxSCC`

```
Input: $\mathcal{ELHIO}$ TBox $\mathcal{T}$, $\mathcal{ELHIO}$ EBox $\mathcal{E}$
Output: $\mathcal{ELHIO}$ TBox $\mathcal{T}$ without extensionally implied SCCs
1 repeat
2     forall the $(C : component) \in SCCs(dlgraph(\mathcal{T}))$ do
3         if $incomingConnections(C) = 0 \wedge \forall \psi \in C.\mathcal{E} \models \psi$ then
4             $\mathcal{T} = \mathcal{T} \backslash C$
5 until Fixpoint
6 return $\mathcal{T}$
```

$dlgraph(\mathcal{T})$. We formalise in Section 4 (Proposition 1) the principles on which we remove this type of SCCs.

Then, through the function $saturate$ [5], Algorithm 1 computes a deductive closure of the EBox, except for recursive predicates: a reduced set of recursive predicates is excluded from the inference to make the saturation process finite. This reduced set is computed by Algorithm 4, where $count(p, \Lambda) = card(\{\lambda \in \Lambda \mid p \in \lambda\})$ and $LoopsIn(\Gamma)$ finds the loops of clauses in the given set of clauses that produce infinite property paths. Excluding some predicates from the inference limits the effect of the EBox in the reduction of the rewritten queries and the possible unfolding of these queries. This limited effect of the EBox means some redundant answers can be produced, which has obviously no effect on the correctness of the answers.

General Algorithm. The result of the preprocessing stage is then used by the general kyrie2 algorithm (Algorithm 3). This algorithm preserves the same stages and optimisations of kyrie to obtain the Datalog program and the unfolding. The main difference with kyrie is the call to the function `useEBox` (Algorithm 5).

Pruning the Rewriting with the EBox. The `useEBox` function, defined by Algorithm 5, uses the EBox to reduce a Datalog program. This can be done by removing clauses or by replacing some of the clauses with other shorter ones. This algorithm performs a set of stages iteratively to reduce the Datalog program considered, until a fixpoint is reached. These stages are:

- Predicates with no extension ($p \sqsubseteq \bot$ in the EBox) are removed, if possible, after saturating the inferences where they participate. A reduced set of recursive

Algorithm 3. General kyrie2 algorithm

```
**Input**: TBox $\Sigma$, EBox $E$, recursive predicates in $\Sigma$ ($R_\Sigma$), recursive predicates in $E$ ($R_E$), UCQ $q$, working mode $mode \in \{\texttt{Datalog}, \texttt{UCQ}\}$
**Output**: Rewritten query $q_\Sigma$
1 $q = removeSubsumed(condensate(q))$
2 $\Sigma_r = reachable(\Sigma, q)$
3 $\Sigma_q = saturate(\texttt{s}, sfRQR, q, \Sigma_r)$
4 $\Sigma_q = \texttt{useEBox}(\Sigma_q, E, R_\Sigma, R_E)$
5 if $mode = \texttt{Datalog}$ then return $\Sigma_q$ $\Sigma_q = \{q_i \in \Sigma_q \mid head(q_i) \neq head(q)\}$
6 $\Sigma_q = \{q_i \in \Sigma_q \mid head(q_i) = head(q)\}$
7 $\Sigma_q = saturate(\texttt{u}, sfNonRec(R_\Sigma), \Sigma_q, \Sigma_q)$
8 $\Sigma_q = \texttt{useEBox}(\Sigma_q, E, R_\Sigma, R_E)$
9 return $\Sigma_q$
```

Algorithm 4. Find reduced sets of recursive predicates: `reducedRecursiveSet`

```
**Input**: Datalog program $\Sigma_q$, EBox $E$
**Output**: Recursive predicates in $\Sigma_q$ ($R_{\Sigma_q}$), recursive predicates in $E$ ($R_E$)
1  $\Lambda_{\Sigma_q} = \texttt{loopsIn}(\Sigma_q)$
2  $\Lambda_E = \texttt{loopsIn}(E)$
3  $R_{\Sigma_q} = \emptyset$
4  $R_E = \emptyset$
5  while $\Lambda_{\Sigma_q} \neq \emptyset \wedge \Lambda_E \neq \emptyset$ do
6      if $\Lambda_{\Sigma_q} \cap \Lambda_E \neq \emptyset$ then
7          $p = p_i \in \Lambda_{\Sigma_q} \cap \Lambda_E / count(p_i, \Lambda_{\Sigma_q}) + count(p_i, \Lambda_E) =$
           $max(count(p_j, \Lambda_{\Sigma_q}) + count(p_j, \Lambda_E)) \forall p_j \in \Lambda_{\Sigma_q} \cap \Lambda_E$
8          $R_{\Sigma_q} = R_{\Sigma_q} \cup \{p\}$
9          $R_E = R_E \cup \{p\}$
10         $\Lambda_{\Sigma_q} = \Lambda_{\Sigma_q} \setminus \{\lambda \in \Lambda_{\Sigma_q} \mid p \in \lambda\}$
11         $\Lambda_E = \Lambda_E \setminus \{\lambda \in \Lambda_E \mid p \in \lambda\}$
12     else
13         if $\Lambda_{\Sigma_q} \neq \emptyset$ then
14             $p = p_i \in \Lambda_{\Sigma_q} / count(p_i, \Lambda_{\Sigma_q}) = max(count(p_j, \Lambda_{\Sigma_q})) \forall p_j \in$
               $\Lambda_{\Sigma_q}$
15             $R_{\Sigma_q} = R_{\Sigma_q} \cup \{p\}$
16             $\Lambda_{\Sigma_q} = \Lambda_{\Sigma_q} \setminus \{\lambda \in \Lambda_{\Sigma_q} \mid p \in \lambda\}$
17         if $\Lambda_E \neq \emptyset$ then
18             $p = p_i \in \Lambda_E / count(p_i, \Lambda_E) = max(count(p_j, \Lambda_E)) \forall p_j \in \Lambda_E$
19             $R_E = R_E \cup \{p\}$
20             $\Lambda_E = \Lambda_E \setminus \{\lambda \in \Lambda_E \mid p \in \lambda\}$
21 return $\langle R_{\Sigma_q}, R_E \rangle$
```

predicates (selected according to Algorithm 4) needs to be kept. We formalise the conditions for removal in Proposition 4.

- Clauses whose answers are subsumed by other clauses are removed. The subsumption of the answers according to the algorithm is formalised in Proposition 5 and therefore they are redundant, as Proposition 6 states.

Algorithm 5. Prune Datalog program Σ_q: `useEBox`

```
Input: EBox $E$, TBox $\Sigma_q$, recursive predicates in $E$ ($R_E$), recursive predicates
       in $\Sigma_q$ ($R_{\Sigma_q}$)
Output: Pruned TBox program $\Sigma_q$
1  repeat
2      $P_e = \{p_i \mid p_i \in predicates(\Sigma_q) \wedge (p_i \sqsubseteq \bot) \in E \wedge p_i \notin R_{\Sigma_q}\}$
3      $\Sigma_q = saturate(\mathtt{u}, \mathtt{sfSel}(P_e), \Sigma_q, \emptyset)$
4      $E_e = \{\gamma_i \in E \mid \forall p \in \gamma_i.\forall \gamma_j \in \Sigma_q.p \notin head(\gamma_j)\}$
5      $E_{\Sigma_q} = \emptyset$
6      forall the clauses $\gamma_1 \in \Sigma_q \cup E_{\Sigma_q}$ do
7          forall the clauses $\gamma_2 \in E_e$ do
8              $\Gamma = \mathtt{resolve}(\gamma_1, \gamma_2, \mathtt{sfNoRec}(R_E))$
9              forall the $\gamma_i \in \Gamma, \gamma_i \neq \gamma_1$ do
10                 forall the $\gamma_j \in \Sigma_q$ do
11                     if $subsumes(\gamma_i, \gamma_j)$ then
12                         $\Sigma_q = \Sigma_q \backslash \{\gamma_j\}$
13                         if $\neg subsumes(\gamma_j, \gamma_i)$ then
14                             $\Sigma_q = \Sigma_q \cup \{\gamma_i\}$
15             $E_{\Sigma_q} = E_{\Sigma_q} \cup \Gamma$
16     forall the $C \in stronglyConnectedComponents(cgraph(\Sigma_q))$ do
17         if $(incomingConnections(C) = 0 \wedge \forall \gamma \in C.\gamma \in E_e$ then
18             $\Sigma_q = \Sigma_q \backslash C$
19     $\Sigma_q = reachable(\Sigma_q)$
20 until Fixpoint
21 return $\Sigma_q$
```

- Clauses where an atom subsumes another atom are condensed. We use resolution to find the condensed version of the clause, which subsumes the original clause. Due to propositions 2 and 7 we know that we can keep any of both clauses. We keep the condensed version, the subsuming clause, since this clause will more likely subsume some other clauses.
- SCCs that are implied by the EBox and receive no connections are removed. This is done again according to Proposition 3.

4 Formalisation

In this section we provide a formalisation for the operations of our algorithms. In particular, we formalise the optimisations of the original kyrie algorithm presented in Section 3. For an easier explanation we introduce two definitions: the contributions of a clause ($\varphi_\Gamma(\gamma)$) and the values for a predicate ($\upsilon_\Gamma(p)$). We recall we denote γ_a subsumes γ_b with $\gamma_a \succeq_s \gamma_b$.

Definition 3. $\varphi_\Gamma(\gamma)$ ***Contributions of a clause*** γ ***in a OBDA system*** $\Gamma = \langle \Sigma, \mathcal{A} \rangle$. *Let* p *be the predicate in the head of* γ*, we define the contributions of* γ *on* Γ *as the set* $\varphi_\Gamma(\gamma) = \{t_1, \ldots, t_n \mid \exists \mu.(\Gamma \models \mu\ body(\gamma)) \wedge (\mu\ head(\gamma) = p(t_1, \ldots, t_n))\}$ *where* μ *is a substitution of the variables in* γ *with the constants* $t_1, \ldots, t_n$*. Please note that*

$\Gamma \models \mu\ body(\gamma)$ means that $\Sigma \cup \mathcal{A} \models \mu\ body(\gamma)$, i.e. the values for the contribution may be implied by other clauses in Σ.

Definition 4. $\upsilon_\Gamma(p)$ ***Values for a predicate*** p ***on*** Γ. *For a given OBDA system $\langle\Sigma, \mathcal{A}\rangle = \Gamma$, we define as the values for a predicate $p \in \Gamma$ the set $\upsilon_\Gamma(p) = \{t_1, \ldots, t_n \mid \Gamma \models p(t_1, \ldots, t_n)\}$.*

Moreover, the values for a predicate p on Γ, $\upsilon_\Gamma(p)$ are divided into the extensional values ($\upsilon^e_\Gamma(p)$) and the intensional values ($\upsilon^i_\Gamma(p)$), so that $\upsilon_\Gamma(p) = \upsilon^e_\Gamma(p) \cup \upsilon^i_\Gamma(p)$ where $\upsilon^e_\Gamma(p) = \{t_1, \ldots, t_n \mid \mathcal{A} \models p(t_1, \ldots, t_n)\}$ and $\upsilon^i_\Gamma(p) = \{t_1, \ldots, t_n \mid \exists\gamma, \mu.\gamma \in \Gamma \wedge \mu\ head(\gamma) = p(t_1, \ldots, t_n) \wedge t_1, \ldots, t_n \in \varphi_\Gamma(\gamma)\}$ where μ is the most general unifier (MGU) applied to $head(\gamma)$, from the variables in γ to the constants in $t_1, \ldots, t_n$.

Informally, the intensional values for a predicate p are the contributions of the clauses where p is in the head, while the contributions of a clause are a projection and selection of the values for the predicates in its body.

For instance, consider the example in Section 2 and more specifically two of the clauses that can be extracted from its axioms:

γ_1: `GradStudent(x) :- MasterStudent(x)` and

γ_2: `GradStudent(x) :- PhDStudent(x)`

We keep the ABox as in the example in Section 2. In this example, the extensional values for `GradStudent` on our system Γ ($\upsilon^e_\Gamma(GradStudent)$) are Ben, Don and Ed. The intensional values for `GradStudent` on our system Γ ($\upsilon^i_\Gamma(GradStudent)$) are Ben, Cal and Ed. The values for the predicate p are therefore the union of both sets: Ben, Cal, Don and Ed. The contributions from clause γ_1 ($\varphi_\Gamma(\gamma_1)$) are Ben and Cal. The contributions from clause γ_2 ($\varphi_\Gamma(\gamma_2)$) are just Don.

Proposition 1. *Let $\mathcal{A}$ be an ABox, Σ and Σ' be two sets of Datalog clauses such that $\Sigma' = \Sigma \cup \{\gamma_r\}$ with p_r as the predicate in the head of γ_r. For the two corresponding OBDA systems $\Gamma \equiv \langle\Sigma, \mathcal{A}\rangle$ and $\Gamma' \equiv \langle\Sigma', \mathcal{A}\rangle$ if $\varphi_{\Gamma'}(\gamma_r) \subseteq \upsilon_\Gamma(p_r)$ then $\upsilon_\Gamma(p) = \upsilon_{\Gamma'}(p)$ for every predicate p in Σ.*

Proposition 2. *Let $\Gamma = \langle\Sigma, \mathcal{A}\rangle$ be a OBDA system and let γ_a, γ_b be two clauses in Σ such that $\gamma_a \succeq_s \gamma_b$. Then, $\varphi_\Gamma(\gamma_b) \subseteq \varphi_\Gamma(\gamma_a)$. For $\Gamma' = \Gamma \setminus \{\gamma_b\}$ holds that $\Phi^q_\Gamma = \Phi^q_{\Gamma'}$.*

We also know that clauses γ_r in the previous context such that $\varphi_{\Gamma \cup \{\gamma_r\}}(\gamma_r) \subseteq \upsilon_{\Gamma \setminus \{\gamma_r\}}(p_r)$ can be added or removed safely from the OBDA system Γ

Proposition 3. *Let $\Gamma = \langle\Sigma, \mathcal{A}\rangle$ be an OBDA system, let E be an EBox satisfied by $\mathcal{A}$ and $cgraph(\Sigma) = \{V_\Sigma, W_\Sigma\}$. Let C be a set of clauses in Σ such that $cgraph(C)$ is a SCC in $cgraph(\Sigma)$ that receives no connections from other SCCs ($\forall(\gamma_a, \gamma_b) \in W.\gamma_b \in cgraph(C) \rightarrow \gamma_a \in cgraph(C)$), and $\forall\gamma \in V.E \models \gamma$. Then, $\Phi^q_{\langle\Sigma,\mathcal{A}\rangle} = \Phi^q_{\langle\Sigma \setminus C,\mathcal{A}\rangle}$.*

The removal of SCCs for $dlgraph(\mathcal{T})$ is analogous to the previously described removal of SCCs for $cgraph(\Sigma)$.

Proposition 4. *Let $\Gamma = \langle\Sigma, \mathcal{A}\rangle$ be an OBDA system where Σ is a set of Datalog clauses, let q be a query, let $\Gamma_q = q \cup \Gamma$ be a Datalog program and let P be a set of non-recursive predicates in Γ_q that have no extensional values, i.e. $\forall p \in P.\upsilon^e_{\Gamma_q}(p) = \emptyset$*

and no $p \in P$ is the query predicate. Let Γ_b be the set of clauses in Γ_q such that $\forall\gamma \in \Gamma_q.\exists p \in P.p \in body(\gamma)$ and let Γ_h be the set of clauses in Γ_q such that $\forall\gamma \in \Gamma_q.\exists p \in P.p \in head(\gamma)$. Let Γ_r be the set of clauses that do not contain p and are generated through resolution from Γ_b and Γ_h with a selection function that selects the predicates $p \in P$. Let Γ_s be the set of clauses $\{\gamma \in \Gamma_q \cup \Gamma_r \mid \gamma \notin \Gamma_h \cup \Gamma_b\}$. Then, $\Phi^q_\Gamma = \Phi^q_{\Gamma_q \cup \Gamma_r} = \Phi^q_{\Gamma_s}$.

Proposition 5. *Let $\Gamma = \langle \Sigma, \mathcal{A} \rangle$ be an OBDA system, let E be an EBox such that $\mathcal{A}$ satisfies E and let E_e the part of the EBox E that contains only predicates with no intensional definition in Σ. For every pair γ, γ_r such that $\gamma \in \Sigma$ and $E_e \cup \gamma \vdash \gamma_r$ then $\varphi_\Gamma(\gamma_r) \subseteq \varphi_\Gamma(\gamma)$.*

Proposition 6. *Let $\Gamma = \langle \Sigma, \mathcal{A} \rangle$, E and E_e be defined as in Proposition 5. Then, for every pair γ, γ_r such that $\gamma \in \Sigma$ and $E_e \cup \gamma \vdash \gamma_r$, and for every query q, $\Phi^q_\Gamma = \Phi^q_{\Gamma'}$, where $\Gamma' = \Gamma \backslash \{\gamma_r\}$.*

Proposition 7. *Let $\Gamma = \langle \Sigma, \mathcal{A} \rangle$, E and E_e be defined as in Proposition 5. Then, for every triple $\gamma, \gamma_r, \gamma_s$ such that $(\gamma, \gamma_s \in \Sigma) \wedge (E_e \cup \gamma \vdash \gamma_r) \wedge (\gamma_r \succeq_s \gamma_s)$, and for every query q, $\Phi^q_\Gamma = \Phi^q_{\Gamma'}$, where $\Gamma' = (\{\gamma_r\} \cup \Gamma) \backslash \{\gamma_s\}$.*

5 Evaluation

Having formalised the proposal, we have performed an empirical evaluation to check our query rewriting optimisations. There is no benchmark in the state of the art to test query rewriting with EBoxes, therefore we have decided to use some of the most widely used ontologies for the evaluation of query rewriting systems [6], in particular we used: Several real world ontologies used in independent projects like Adolena (A), Vicodi (V) and StockExchange (S). Benchmark ontologies, like a DL-Lite$_R$ version of LUBM (U). Artificial ontologies to test the impact of property paths, path1 and path5 (P1, P5). Previous ontologies with auxiliary predicates for DL-Lite compliance [9] (UX, P5X). Additionally, we consider the extension of previous ontologies with axioms in $\mathcal{ELHIO}$ and beyond DL-Lite (UXE, P5XE).

We have have expanded previous assets with a set of synthetically-generated EBoxes, using a randomized and parametrised algorithm, with parameters:

size: the size of the EBox relative to the size of the TBox: zero is an empty EBox, and one is an EBox with as many axioms as the TBox.

cover: how much of the TBox is covered by the EBox: zero means that all the axioms in the EBox will be randomly generated, one means that all the axioms in the EBox will come from the TBox.

reverse: how many of the axioms obtained from the TBox (cover) are reversed in the EBox (the reverse of $A \sqsubseteq B$ being $B \sqsubseteq A$) wrt the original form in the TBox: zero means that no axioms are reversed, one means that all axioms are reversed. The reversed axioms belong to the cover, i.e. if the cover is zero this number has no effect.

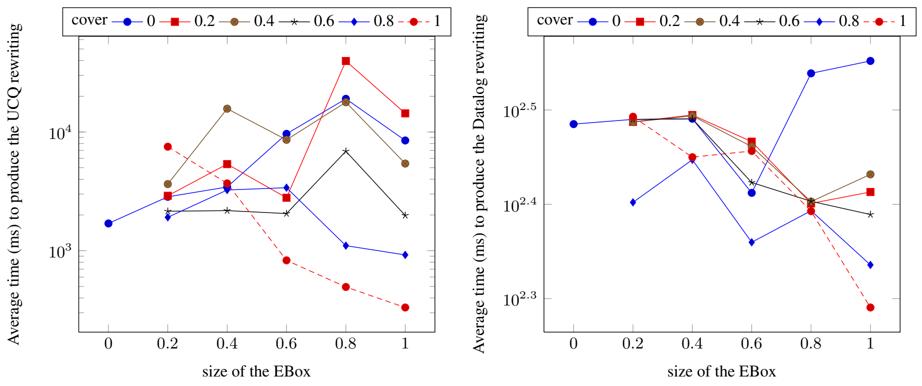
Table 2. Results for ontology V (original size 222 clauses) with EBoxes I, II, III and IV

Query independent information				
EBox	I	II	III	IV
PT	109	2047	24266	2859
PS	222	195	171	111
size	0.0	0.2	0.8	0.8
cover	0.0	0.8	0.2	0.8
rev	0.0	0.0	0.0	0.0
PT:	preprocess time (ms)			
PS:	preprocessed size			

query	Datalog time(ms)				Datalog size				UCQ time(ms)				UCQ size			
	I	II	III	IV	I	II	III	IV	I	II	III	IV	I	II	III	IV
1	0	0	157	235	15	13	14	9	0	0	516	672	15	13	14	9
2	16	16	157	234	10	10	10	10	16	16	500	656	10	10	10	10
3	0	0	125	235	35	30	28	15	31	15	485	813	72	57	54	15
4	0	16	219	188	41	38	22	16	63	94	735	719	185	170	3	42
5	16	16	172	250	8	5	7	1	32	31	609	719	30	9	15	1
6	0	15	234	188	18	14	14	11	0	15	578	641	18	14	14	11
7	0	0	125	172	27	23	23	20	94	125	1359	1359	180	140	140	110

All these parameters can be any real number between zero and one, except for size, which can be greater than one (e.g. a size of two means the EBox is twice the size of the TBox). Of course, the latter parameters are only significant when previous ones are greater than zero. If the cover is less than one then axioms up to one are generated by selecting randomly the LHS or RHS of other rules, adding an axiom $A_1 \sqsubseteq A_2$ for each pair A_1, A_2 found this way. For the A_i that are classes, nothing else is done. If some of them is a property P then the axiom $\exists P$ is added and with a probability of $1/2$ some other LHS or RHS A_3 of some other rule is selected. If A_3 is selected then $\exists P$ is expanded into $\exists P.A_3$. This is repeated recursively until (1) a class is selected or (2) a property is selected and not expanded (expansion probability: $1/2$).

The "size" is the main parameter. It is meant to help to evaluate the impact of the EBox on the results. The "cover" specifies how much of the EBox is related to the TBox and how much is random. It is meant to help to evaluate how the relation between EBox and TBox impacts on the results. The "reverse" specifies how many of the EBox axioms that are obtained from the TBox remain unaltered and how many are reversed. If the axiom is unaltered then the subconcepts are redundant, otherwise the superconcept is redundant. This parameter helps to evaluate the impact of redundancy in these cases.



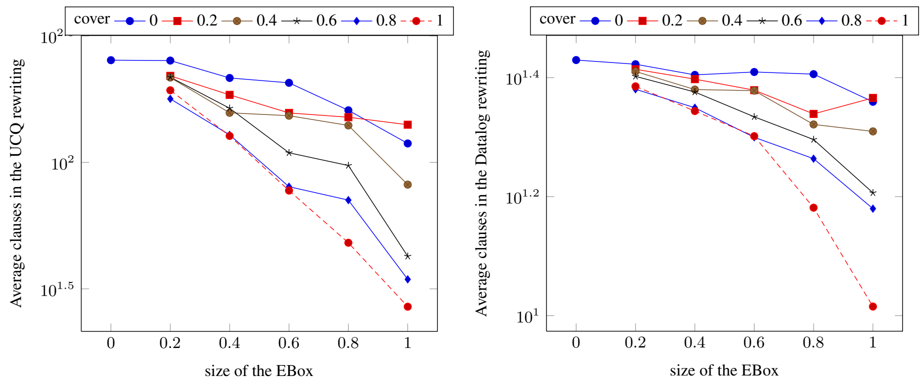
(a) UCQ rewriting times for cover wrt size

(b) Datalog rewriting times for cover wrt size

Fig. 1. Average rewriting times relating size and cover. All queries and ontologies considered. "Reverse" is zero in all cases.

We have run the tests with a set of EBoxes, selecting the values 0.0, 0.2, 0.4, 0.6, 0.8 and 1.0 for each of the parameters described above. When the size is zero all the other parameters are zero, and when cover is zero reverse is zero as well. This involves a total of $1 + 5 * (1 + 5 * 6) = 156$ EBoxes, for each ontology, used for all queries. Due to space limitations, we present a small excerpt of the results for a single ontology in Table 2: the full results for all the ontologies and the EBoxes can be found online[4].

The results have been obtained on cold runs, by restarting the application after every query (passed in the application invocation parameters). The consistency of the results regardless of how the system is run has been ensured by measuring the query rewriting time and discarding operations done before and after it. The hardware used in the evaluation is a Intel®Core™2 6300 @1.86GHz with 2GB of RAM, Windows® XP and Java™version 1.6.0_33, with default settings for the Java Virtual Machine.



(a) number of clauses in the UCQ rewriting for cover wrt size **(b)** number of clauses in the Datalog rewriting for cover wrt size

Fig. 2. Average rewriting sizes relating size and cover. All queries and ontologies considered. “Reverse” is zero in all cases.

With the evaluation results we can observe that:

- In the computation of Datalog rewritings, *using the EBox allows for obtaining equal or smaller Datalog programs for all queries, with negligible effects on the query rewriting time*.
- For UCQ rewritings, our results show that, in general, *the number of clauses is reduced as the EBox increases in size and similarity (”cover”) with the TBox*. This reduction in the number of clauses usually implies a reduction in the time required for the query rewriting process, as we can see in Figure 1. This figure shows the average time to produce a UCQ or a Datalog rewriting and how EBoxes with different sizes and TBox coverage influence this time. More precisely:
  - ○ We can notice that EBoxes with both a high value for cover and size tend to reduce the query rewriting time when compared with other EBoxes or no EBox.

[4] `http://purl.org/net/jmora/extensionalqueryrewriting`

  - When the EBox involves more random axioms (e.g. EBox III in Table 2 with 0.8 for ”size” and 0.2 for ”cover”) the results are less predictable. More specifically, we can see that for UCQ size the EBox III behaves better than EBox IV in query 4 and it behaves worse than EBox II in query 5. This variation depends on the query, the EBox may contain axioms that imply the subsumption of atoms in the query. If these axioms are in the TBox, their presence in the EBox has no impact in the results (EBox IV). However, if these axioms are not in the TBox, this can lead to further clause condensation, with the elimination of subsumed clauses and the ones generated from these. This elimination of clauses means a potentially strong reduction in the size of the results and the time required to produce them.
- Even *in the cases where the query rewriting times are higher*, we can see *an important reduction in the number of clauses generated*, in the Datalog and UCQ rewritings, as Figure 2 shows. This reduction in the number of clauses implies a reduction in the redundancy of the queries that are generated. This simplifies the computation required for answering the query by the other layers of the OBDA system.

6 Conclusions and Future Work

We can conclude from the evaluation that the impact of EBoxes is clearly noticeable and generally positive. This is especially relevant when the EBox is similar to the TBox in size and contents, which may imply a reduction in query rewriting time. An EBox that is randomly generated can have a strong positive impact in query rewriting time if it implies subsumption between the atoms in the query. Even for EBoxes that increase the query rewriting time, the reduction of redundancy in the generated queries and answers should produce an improvement in the execution of these queries.

Among the possibilities for future work we consider the extension to Datalog± [15]. The Datalog± family of languages provides interesting opportunities to explore the expressiveness that can be achieved while dealing with recursion [16].

Another line to explore in the future is considering only the part of the EBox that is similar to the TBox, which is guaranteed to have a positive impact. For example, the axioms or clauses in the EBox that are not related with the TBox could be discarded during the preprocessing. By doing this, the results would be more predictable and the impact of a large EBox not related with a TBox would be the same as for a small EBox that is very related to the TBox.

We also consider exploring whether it is possible to extend the input and output query language to SPARQL. The expressiveness of SPARQL 1.1 allows for a limited recursion, e.g., property paths. The possibility of using subqueries is also interesting to limit the combinatorial explosion implied by the unfolding.

Finally, we plan to experiment our technique on some real-world use cases [17], which would provide information about the relation between the TBox and the corresponding ABox and EBox. This would allow further optimisations that would address specific characteristics of the EBoxes that are more usual in the use cases. With populated ABoxes, obtaining the answers to the queries would allow a better quantification and evaluation of the impact in the whole query answering process.

Acknowledgements. The work presented in this paper has been funded by a PIF grant (Personal Investigador en Formación) from UPM (RR01/2008) and by the EU FP7 project Optique – Scalable End-user Access to Big Data (grant n. FP7-318338).

References

1. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. In: Spaccapietra, S. (ed.) Journal on Data Semantics X. LNCS, vol. 4900, pp. 133–173. Springer, Heidelberg (2008)
2. Rodríguez-Muro, M., Calvanese, D.: Dependencies: Making ontology based data access work in practice. In: Alberto Mendelzon Workshop on Foundations of Data Management (2011)
3. Rosati, R.: Prexto: Query rewriting under extensional constraints in DL $-$ *lite*. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, O., Presutti, V. (eds.) ESWC 2012. LNCS, vol. 7295, pp. 360–374. Springer, Heidelberg (2012)
4. Console, M., Lenzerini, M., Mancini, R., Rosati, R., Ruzzi, M.: Synthesizing extensional constraints in ontology-based data access. In: Description Logics, pp. 628–639 (2013)
5. Mora, J., Corcho, O.: Engineering optimisations in query rewriting for OBDA. In: I-SEMANTICS 2013, Austria, pp. 41–48. ACM (2013)
6. Mora, J., Corcho, O.: Towards a systematic benchmarking of ontology-based query rewriting systems. In: Alani, H., et al. (eds.) ISWC 2013, Part II. LNCS, vol. 8219, pp. 376–391. Springer, Heidelberg (2013)
7. Pérez-Urbina, H., Motik, B., Horrocks, I.: Tractable query answering and rewriting under description logic constraints. Journal of Applied Logic, 186–209 (2010)
8. Calvanese, D., Lembo, D., Lenzerini, M., Poggi, A., Rosati, R.: Ontology-based database access. Technical report, CiteSeerX (2007)
9. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. Journal of Automated Reasoning 39(3), 385–429 (2007)
10. Gottlob, G., Orsi, G., Pieris, A.: Ontological queries: Rewriting and optimization (extended version). arXiv:1112.0343 (December 2011)
11. Rosati, R., Almatelli, A.: Improving query answering over DL-Lite ontologies. In: Principles of Knowledge Representation and Reasoning. AAAI Press (2010)
12. Chortaras, A., Trivela, D., Stamou, G.: Optimized query rewriting for OWL 2 QL. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS, vol. 6803, pp. 192–206. Springer, Heidelberg (2011)
13. Venetis, T., Stoilos, G., Stamou, G.: Query extensions and incremental query rewriting for OWL 2 QL ontologies. Journal on Data Semantics (2013)
14. Eiter, T., Ortiz, M., Simkus, M., Tran, T.K., Xiao, G.: Query rewriting for Horn-SHIQ plus rules. In: 26th AAAI Conference on Artificial Intelligence (2012)
15. Calì, A., Gottlob, G., Lukasiewicz, T.: A general datalog-based framework for tractable query answering over ontologies. Journal of Web Semantics (2012)
16. Cadoli, M., Palopoli, L., Lenzerini, M.: Datalog and description logics: Expressive power. In: Cluet, S., Hull, R. (eds.) DBPL 1997. LNCS, vol. 1369, pp. 281–298. Springer, Heidelberg (1998)
17. Priyatna, F., Corcho, O., Sequeda, J.F.: Formalisation and experiences of R2RML-based SPARQL to SQL query translation using Morph. In: International World Wide Web Conference, International World Wide Web Conferences Steering Committee (2014)