OWL Query Answering based on Query Extension

Birte Glimm[1], Yevgeny Kazakov[1], Ilianna Kollia[2], and Giorgos Stamou[2]

[1] University of Ulm, Germany, <firstname.surname>@uni-ulm.de
[2] National Technical University of Athens, Greece, ilianna2@mail.ntua.gr, gstam@cs.ntua.gr

Abstract. The paper presents an approach for optimizing query answering algorithms that are based on approximate instance retrieval. We consider SPARQL instance queries over OWL ontologies and use the OWL 2 Direct Semantics entailment regime of SPARQL for their evaluation. Approximate query answering algorithms are based on the creation of two sets; the set of certain or known query answers and the set of possible query answers, which require checks to determine whether they are real answers. Typically, it is expensive to check the possible answers hence our goal in this paper is to reduce the number of possible answers returned by approximate reasoning algorithms. We present an approach for using schema knowledge from the terminology (TBox) to optimize the evaluation of SPARQL instance queries. We proceed by transforming the query into a set of assertions (ABox). We then show how the TBox and this (small) query ABox can be used to build an equivalent query where the additional query atoms can be used for reducing the set of possible mappings for query variables.

1 Introduction

Query answering—the computation of answers to users' queries w.r.t. ontologies and data—is an important task in the context of the Semantic Web that is provided by many OWL reasoners. Although much effort has been spent on optimizing the 'reasoning' part of query answering, i.e., the extraction of the individuals that are instances of a class or property, less attention has been given to optimizing the actual query answering part when ontologies in expressive languages are used. The SPARQL query language [10], which was standardized in 2008 by the World Wide Web Consortium (W3C), is widely used for expressing queries in the context of the Semantic Web. We use the OWL Direct Semantics entailment regime of SPARQL 1.1 [4] according to which RDF triples from basic graph patterns are first mapped to extended OWL axioms which can have variables in place of classes, properties and individuals and are then evaluated according to the OWL entailment relation. We focus only on queries with variables in place of individuals since such queries are very common. We call the extended OWL axioms *query atoms* or *atoms*.

Evaluating queries over OWL 2 DL ontologies using approximate query answering algorithms usually involves performing expensive consistency checks for deciding whether possible answers are real. For example, the description logic $\mathcal{SROIQ}$, which underpins the OWL 2 DL standard has a worst case complexity of 2-NExpTime. In this paper we focus on optimizing such query answering algorithms that are based on approximate instance retrieval. We first define what an approximate query answering

algorithm is and give a simple query answering algorithm that is directly based on approximate instance retrieval. We afterwards present an efficient algorithm that is based on a technique called *query extension*. According to query extension, for a given query q, we compute an equivalent query $\hat{q}$ that can be evaluated more efficiently. We first replace the variables in q with fresh individual names and we afterwards perform realization, i.e., we materialize entailed class and property assertions, for the queried TBox and (small) query ABox. Replacing the individual names again with the corresponding variable names then yields $\hat{q}$. The additional query atoms in $\hat{q}$ can then be used for reducing the set of possible mappings for query variables. We provide a prototypical implementation and evaluation of the proposed optimization, which shows that it can lead to an improvement of up to two orders of magnitude in the query execution times.

The results in this paper have partly been published at the International Description Logic Workshop 2013 [3], but only in this paper an empirical evaluation of the proposed query optimization is performed.

2 Preliminaries

In this section we give a brief introduction to the SPARQL instance queries, which we use throughout the paper. Instead of OWL syntax, for brevity, we use the description logic (DL) [1] syntax for examples. We assume that an ontology O is a pair $\langle \mathcal{T}, \mathcal{A} \rangle$ with $\mathcal{T}$ a TBox that also includes axioms involving properties and $\mathcal{A}$ an ABox.

The WHERE clause of a SPARQL query consists of graph patterns. Basic graph patterns (BGPs) can be composed to more complex patterns using operators such as UNION and OPTIONAL for alternative and optional selection criteria. The evaluation of (complex) graph patterns is done by evaluating each BGP separately and combining the results of the evaluation. We only consider the evaluation of BGPs since this is the only thing that is specific to a SPARQL entailment regime. We further focus on SPARQL instance queries, i.e., BGPs that retrieve tuples of individuals, which are instances of the queried class expressions and properties. Such BGPs are first mapped to OWL class and (object) property assertions that allow for variables in place of individuals. For further details, we refer interested readers to the W3C specification that defines the mapping between OWL structural objects and RDF graphs [9] and to the specification of the OWL Direct Semantics entailment regime of SPARQL [4] that defines the extension of this mapping between BGPs and OWL objects with variables. For brevity, we directly write mapped BGPs in DL syntax extended to allow for individual variables.

Definition 1 (Query). *A* query signature $\mathcal{S}$ *is a four-tuple* (N_C, N_R, N_I, V)*, where the tuple* (N_C, N_R, N_I) *is a signature and V is a countable, infinite set of (individual) variables disjoint from* N_C*,* N_R*, and* N_I*. A* term *is an element from* $N_I \cup V$*. Let C be an OWL 2 DL class expression, t, t′ terms. An atom is an expression of the form* $C(t)$ *(class atom) or* $r(t, t')$ *(property atom). A* query *q is a formula* $\vec{x} \leftarrow \mathsf{at}_1, \ldots, \mathsf{at}_n$*, where* $\mathsf{at}_1, \ldots, \mathsf{at}_n$ *are atoms and* $\vec{x}$ *is the tuple of variables contained in* $\mathsf{at}_1, \ldots, \mathsf{at}_n$*. We use* $Var(q)$ ($Var(\mathsf{at})$ *for a query atom* at*) to denote the set of variables in q (*at*), respectively. We write* $|q|$ *to denote the number of axiom templates in q.*

A query mapping *for q, short just mapping, is a total function* $\mu\colon Var(q) \to N_I$. *We use* $dom(\mu)$ *to denote the* domain *of* μ. *We write* $\mu(\mathsf{at})$ $(\mu(q))$ *to denote the result of replacing each variable x in* at (q) *with* $\mu(x)$. *A query mapping* μ *is a* certain answer *for q over an ontology* $\mathcal{O}$, *written* $\mathcal{O}, \mu \models q$, *if* $\mathcal{O} \models \mu(\mathsf{at})$ *for each atom* at *in q. We denote the set of all certain answers for q over* $\mathcal{O}$ *with* $\mathsf{ans}(\mathcal{O}, q)$.

Let $X = \{x_1, \ldots, x_n\}$ *be a set of variables and M a set of query mappings. The* projection of X over M *is the set* $M_{|X} = \{\{x_1 \mapsto \mu(x_1), \ldots, x_n \mapsto \mu(x_n)\} \mid \mu \in M\}$.

For a query $q = \vec{x} \leftarrow \mathsf{at}_1, \ldots, \mathsf{at}_n$, we often write q just as a set of atoms $\{\mathsf{at}_1, \ldots, \mathsf{at}_n\}$, when the order of atoms and the order of the variables in $\vec{x}$ is not important. In the following we use A for a class, C for a class expression, r for an object property, a, b for individuals and x, y for variables.

3 Query Answering via Approximate Instance Retrieval

In this section, we present a technique that uses approximate query answering algorithms in order to optimize the evaluation of queries. Approximate query answering algorithms can either be sound and incomplete, i.e., they under-approximate the set of certain or known answers or they can be complete but unsound, i.e., they over-approximate the set of certain answers. Typical examples of such algorithms rewrite a knowledge base into a simpler logic in such a way that computing the results over the simplified knowledge base yields the desired under- or over-approximation [7, 12]. Another possibility is to use a pre-model or complete and clash-free tableau generated by an OWL reasoner from which one can then read-off certain instances of classes and properties by analyzing which class and property facts have been added deterministically to the pre-model, i.e., one can obtain an under-approximation for class and property instances. Similarly, one can analyze the non-deterministically added and absent class and property facts to compute an over-approximation. More formally, we define an approximate query answering algorithm as follows:

Definition 2 (Approximate Query Answering Algorithm). *Let* $\mathcal{O}$ *be an ontology and q a query. An* approximate query answering algorithm $\mathsf{apprQA}(\mathcal{O}, q)$ *returns a pair of sets* $\langle K[q], P[q]\rangle$ *of query mappings for q such that:*

1. *for each* $\mu \in K[q]$, $\mu \in \mathsf{ans}(\mathcal{O}, q)$, *and*
2. *for each* $\mu \in \mathsf{ans}(\mathcal{O}, q)$, $\mu \in K[q] \cup P[q]$.

We call $K[q]$ *the known and* $P[q]$ *the possible answers for q over* $\mathcal{O}$.

An approximate query answering algorithm is called an approximate instance retrieval algorithm *if the input query can only consist of a single query atom. For brevity, we write* $\mathsf{inst}(\mathcal{O}, \mathsf{at})$ *for the call of such an algorithm.*

Without loss of generality, in the rest of the paper we assume that $K[\cdot] \cap P[\cdot] = \emptyset$ holds for every approximate query answering algorithm.

For ease of presentation, we often write $K[C] = \{a_1, \ldots, a_n\}$ instead of $K[C(x)] = \{\{x \mapsto a_1\}, \ldots, \{x \mapsto a_n\}\}$ and similarly for property atoms, queries, and possible answers.

While it is well-known [11, 8, 5] how an approximate instance retrieval algorithm can be implemented, it is less clear how one can implement a general approximate query answering algorithm. For SPARQL instance queries, an obvious approach is to use an approximate instance retrieval algorithm for each query atom and then compute the join of the resulting sets $K[\cdot]$ and $P[\cdot]$, where we straightforwardly interpret the mappings as relations. The following example illustrates that some possible answers can easily be rejected.

Example 1. Let $\mathcal{O}$ be an ontology and q the query $\langle x, y\rangle \leftarrow C(x), r(x, y), D(y)$. Suppose that (possibly as a result of $\mathsf{inst}(\mathcal{O}, C(x))$, $\mathsf{inst}(\mathcal{O}, r(x, y))$ and $\mathsf{inst}(\mathcal{O}, D(y))$) we have

$$K[C] = \{a\} \qquad K[r] = \{\langle a, c\rangle\} \qquad K[D] = \{c\}$$
$$P[C] = \{b\} \qquad P[r] = \{\langle b, d\rangle, \langle b, e\rangle\} \qquad P[D] = \{d\}$$

Even if we do not know $\mathcal{O}$, we can conclude that $\langle a, c\rangle$ is a certain answer to q, since $a \in K[C]$, $\langle a, c\rangle \in K[r]$ and $c \in K[D]$. However, only $\langle b, d\rangle$ is a possible answer for q since $b \in P[C]$, $\langle b, d\rangle \in P[r]$ and $d \in P[D]$; $\langle b, e\rangle$ cannot be an answer for q since although $\langle b, e\rangle \in P[r]$ and $b \in P[C]$, $e \notin K[D] \cup P[D]$.

Algorithm intersecQans (see Algorithm 1) formalizes this idea, which we also illustrate in the next example.

Example 2. Let q be as in Example 1. After possibly bringing the query atoms into a beneficial execution order and initializing $K[C]$ and $P[C]$, the sets $K[q]$ and $P[q]$ are initialized to contain partial mappings that only become query mappings once the algorithm is finished. During the following iterations, the mappings in $K[q]$ are extended by performing a natural join with the known answers for the current atom at. The set $P[q]$ is extended by performing a natural join of $P[q]$ with both $K[\mathsf{at}]$ and $P[\mathsf{at}]$ and of $K[q]$ with $P[\mathsf{at}]$. For the example query, we next process $r(x, y)$ and obtain $K[q] = \{\langle a, c\rangle\}$ and $P[q] = \{\langle b, d\rangle, \langle b, e\rangle\}$. We finally process $D(y)$ and keep $K[q] = \{\langle a, c\rangle\}$ and $P[q] = \{\langle b, d\rangle\}$.

Lemma 1. *Algorithm* intersecQans *is an approximate query answering algorithm for SPARQL instance queries.*

Proof (sketch). The lemma can straightforwardly be shown by induction on the length of the input query.

The method order in intersecQans is optional and can use query ordering techniques from databases to order the query atoms based on the cardinalities of the sets $K[\mathsf{at}]$ and $P[\mathsf{at}]$, e.g., one would prefer joins over connected atoms and join a small relation (an atom at with smaller $K[\mathsf{at}]$ and $P[\mathsf{at}]$ sets) with a bigger one where possible. Algorithm 2 shows how we can evaluate SPARQL instance queries using intersecQans.

4 Query Extension

The question now is: given an ontology $\mathcal{O}$ and a query q, how we can find a more efficient way to further reduce the cardinality of $P[q]$ with the aid of inst.

Algorithm 1 intersecQans$(\mathcal{O}, q)$

Require: $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$: an OWL 2 DL ontology
q: a query over $\mathcal{O}$
Ensure: $\langle K[q], P[q] \rangle$: $K[q], P[q]$ sets of known and possible answers for q over $\mathcal{O}$

$\mathsf{at}_1, \ldots, \mathsf{at}_n := \mathsf{order}(q, \mathcal{O})$
for $i = 1, \ldots, n$ **do**
  $\langle K[\mathsf{at}_i], P[\mathsf{at}_i] \rangle := \mathsf{inst}(\mathcal{O}, \mathsf{at}_i)$
  if $K[q]$ and $P[q]$ not initialized **then**
    $\langle K[q], P[q] \rangle := \langle K[\mathsf{at}_i], P[\mathsf{at}_i] \rangle$
  else
    $K[q] := K[q] \bowtie K[\mathsf{at}_i]$
    $P[q] := (P[q] \bowtie P[\mathsf{at}_i]) \cup (K[q] \bowtie P[at_i]) \cup (P[q] \bowtie K[\mathsf{at}_i])$
  end if
end for
return $\langle K[q], P[q] \rangle$

Algorithm 2 evaluateIntersecQans$(\mathcal{O}, q)$

Require: $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$: an OWL 2 DL ontology
q: a query over $\mathcal{O}$
Ensure: the certain answers for q over $\mathcal{O}$

$\langle K[q], P[q] \rangle := \mathsf{intersecQans}(\mathcal{O}, q)$
return $\{\mu \mid \mu \in K[q]\} \cup \{\mu \mid \mu \in P[q], (\mathcal{O}, \mu) \models q\}$

Example 3. Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an ontology with $\mathcal{O} \models \exists r.\top \sqcap C \sqsubseteq B$, $q = \{C(x), \exists r.D(x)\}$ a query, $\mathsf{inst}(\mathcal{O}, C(x)) = \langle \{b\}, \{a\} \rangle$ and $\mathsf{inst}(\mathcal{O}, B(x)) = \langle \{a\}, \emptyset \rangle$ and $\mathsf{inst}(\mathcal{O}, \exists r.D(x)) = \langle \emptyset, \{a, b\} \rangle$. From Algorithm 1 we have $K[q] = \emptyset$ and $P[q] = \{a, b\}$. In this case, b is no longer a possible answer of q. If b would be a possible mapping for x, it would have an r-successor (since $\exists r.D(x) \in q$) and it would be an instance of C (since $C(x) \in q$) and, hence, b should be in $K[B] \cup P[B]$ to satisfy the entailed axiom, which is not the case.

The atom $B(x)$ in the above example is called a restricting atom. We now give a definition of restricting atoms that will help us define an efficient algorithm.

Definition 3 (Restricting Atoms). *Let $\mathcal{O}$ be an ontology, q a query,* at *a query atom with $Var(\mathsf{at}) \subseteq Var(q)$, and* inst *an approximate instance retrieval algorithm. Then we say that* at restricts *q if*

$$P[q]_{|Var(\mathsf{at})} \cap (K[\mathsf{at}] \cup P[\mathsf{at}]) \subset P[q]_{|Var(\mathsf{at})}$$

where $\langle K[q], P[q] \rangle = \mathsf{intersecQans}(\mathcal{O}, q)$ and $\langle K[\mathsf{at}], P[\mathsf{at}] \rangle = \mathsf{inst}(\mathcal{O}, \mathsf{at})$.

Example 4. Going back to Example 3, we find that $B(x)$ is indeed a restricting atom for q according to Definition 3 since we have $P[q]_{|\{x\}} = \{a, b\}$ and $P[q]_{|\{x\}} \cap (K[B] \cup P[B]) = \{a\}$, which clearly is a subset of $P[q]_{|\{x\}}$.

If we want to preserve the certain answers of q, we should use restricting atoms that do not change the answers of q. Let q and q' be queries such that $q' = q \cup \{at\}$. If q and

q' are equivalent queries, i.e., q and q' yield the same answers over a fixed TBox and any ABox, and at restricts q, then we can safely prune the set of possible answers for q with the help of at. Obviously, we could also use more than one restricting atom to even further restrict q. Since such atoms are not given as input, we address the problem of (efficiently) computing such restricting atoms within an approximate query answering algorithm after showing that using restricting atoms for queries indeed preserves the certain answers.

Lemma 2. *Let* $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ *be an ontology,* q *and* q' *two queries such that* $q' = q \cup \{\mathsf{at}\}$*,* $\mathsf{ans}(\mathcal{O}, q) = \mathsf{ans}(\mathcal{O}, q')$*,* $\langle K[q], P[q] \rangle = \mathsf{intersecQans}(\mathcal{O}, q)$*,* $\langle K[\mathsf{at}], P[\mathsf{at}] \rangle = \mathsf{inst}(\mathcal{O}, \mathsf{at})$*, and* at *restricts* $P[q]$*.*

1. *An algorithm that returns* $\langle K[q], \{\mu \in P[q] \mid \mu_{|Var(at)} \in (K[at] \cup P[at])\} \rangle$ *given* $\mathcal{O}$ *and* q *as input is an approximate query answering algorithm and*
2. $|\{\mu \in P[q] \mid \mu_{|Var(at)} \in (K[at] \cup P[at])\}| < |P[q]|$.

Proof (Sketch). 1. According to Lemma 1, $\mathsf{intersecQans}(\mathcal{O}, q)$ is an approximate query answering algorithm. Hence, the first condition on approximate query answering algorithms is satisfied. That also the second condition is satisfied can be shown by assuming, to the contrary of what is to be shown, that there is a certain answer μ such that $\mu \notin K[q] \cup \{\mu \in P[q] \mid \mu_{|Var(\mathsf{at})} \in (K[\mathsf{at}] \cup P[\mathsf{at}])\}$, i.e., $\mu_{|Var(\mathsf{at})} \notin K[\mathsf{at}] \cup P[\mathsf{at}]$. Using the definition of restricting atoms, we can then show a contradiction.

2. Since $\{\mu \in P[q] \mid \mu_{|Var(\mathsf{at})} \in (K[\mathsf{at}] \cup P[\mathsf{at}])\}$ is a strict subset of $\{P[q]\}$ by Definition 3 and since at restricts q the claim follows.

In order to define an improved approximate query answering algorithm based on Lemma 2, we need to find a way of computing such restricting atoms. In the following, we will see how we can use the TBox for this aim. We first create a small ABox, called query ABox, from the query atoms by mapping their variables to fresh individuals that do not appear in the query or the TBox to avoid interactions. We then materialize this ABox w.r.t. implicit class and property assertions for the individuals that appear in it and the (possible) additional individuals (nominals) added by the TBox creating the extended query ABox. Afterwards, we can go to the extended query by replacing individuals back with variables of the initial query.

The proposed query extension method is similar to the method for deciding containment between conjunctive queries [2], with the main difference that instead of checking query containment, we construct a query contained in the given query ourselves.

Let Z be a set of axioms or query atoms. In the definition below with ${N_C}^Z$, ${N_R}^Z$ and ${N_I}^Z$ we denote the set of classes, properties and individuals respectively appearing in Z.

Definition 4 (Query Extension). *Let* $\mathcal{T}$ *be a TBox over a signature* $\mathcal{S}$*,* q *a query over the corresponding query signature* $\mathcal{S}_q$*, and* f *a total function from* $Var(q)$ *to a set of individual names from* N_I*. The* query ABox $\mathcal{A}_q^f$ *for* q *w.r.t.* f *is defined as follows:*

$$\mathcal{A}_q^f = \{f(at) \mid at \in q\}$$

The extended query ABox $\hat{\mathcal{A}}_q^{f,\mathcal{T}}$ *for* q *w.r.t.* f *and* $\mathcal{T}$ *is defined as:*

$$\begin{aligned}\hat{\mathcal{A}}_q^{f,\mathcal{T}} = \ & \mathcal{A}_q^f \cup \{B(a) \mid B \in {N_C}^{\mathcal{T} \cup \mathcal{A}_q^f}, a \in {N_I}^{\mathcal{T} \cup \mathcal{A}_q^f} \text{ and } \mathcal{T} \cup \mathcal{A}_q^f \models B(a)\} \\ & \cup \{r(a,b) \mid r \in {N_R}^{\mathcal{T} \cup \mathcal{A}_q^f}, a, b \in {N_I}^{\mathcal{T} \cup \mathcal{A}_q^f} \text{ and } \mathcal{T} \cup \mathcal{A}_q^f \models r(a,b)\}\end{aligned}$$

If $\mathcal{T} \cup \mathcal{A}_q^f$ *is consistent,* f *is additionally bijective and* $range(f) = N_I \setminus {N_I}^{\mathcal{T} \cup q}$*, we can define the* extended query $\hat{q}$ *w.r.t.* f *and* $\hat{\mathcal{A}}_q^{f,\mathcal{T}}$ *as follows:*

$$\hat{q} = \{f^-(at) \mid at \in \hat{\mathcal{A}}_q^{f,\mathcal{T}}\}$$

Example 5. Suppose we have an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ such that $\mathcal{T}$ contains one axiom, i.e., $\exists r.\top \sqcap C \sqsubseteq B$ and $q = \{C(x), \exists r.D(x)\}$ from Example 3. In this case, the query ABox for q is $\mathcal{A}_q^f = \{C(a_x), \exists r.D(a_x)\}$ where f maps the variable x to the individual name a_x. The extended query ABox for q w.r.t. f and $\mathcal{T}$ is $\hat{\mathcal{A}}_q^{f,\mathcal{T}} = \{C(a_x), \exists r.D(a_x), B(a_x)\}$ and the extended query w.r.t $\hat{\mathcal{A}}_q^{f,\mathcal{T}}$ is $\hat{q} = \{C(x), \exists r.D(x), B(x)\}$. Please note that $\hat{q}$ has the same answers as q (w.r.t. any ABox), because $\mathcal{O} \models \exists r.\top \sqcap C \sqsubseteq B$.

Note that we want q and $\hat{q}$ to have the same answers w.r.t. any ABox. For this reason we restrict f to map variables to individuals not appearing in q or $\mathcal{T}$ (nominals). If $\mathcal{T}$ has nominals and we map a query variable to a nominal in $\mathcal{T}$ then $\hat{\mathcal{A}}_q^f$ may contain consequences that come from the interaction of query mappings and TBox nominals. For example, let $\mathcal{T} = \{\exists r.\{b\} \sqsubseteq A\}$ and $q = \{r(x,y), C(y)\}$. If we choose $f = \{x \mapsto a, y \mapsto b\}$ then $\hat{\mathcal{A}}_q^f = \{r(a,b), C(b), A(a)\}$ and $\hat{q} = \{r(x,y), C(y), A(x)\}$. The mapping $\{x \mapsto a, y \mapsto c\}$ is an answer of q, whereas it is not an answer of $\hat{q}$ w.r.t. the ABox $\mathcal{A} = \{r(a,c), C(c)\}$.

Lemma 3. *The extended query* $\hat{q}$ *created as in Definition 4 is unique.*

Proof (Sketch). The lemma follows since the extended query ABoxes produced w.r.t. different functions f are isomorphic to each other, i.e., identical modulo renaming of individuals.

Intuitively (see Example 5), the extended query $\hat{q}$ adds atoms to q that do not change the set of answers of q, which we formalize by Theorem 1.

Theorem 1. *Let* $\mathcal{T}$ *be a TBox,* q *a query, and* $\hat{q}$ *the extended query as in Definition 4. Then,* $\mathcal{T} \models q \equiv \hat{q}$*, i.e., for any ABox* $\mathcal{A}$*,* $\mathsf{ans}(\langle \mathcal{T}, \mathcal{A} \rangle, q) = \mathsf{ans}(\langle \mathcal{T}, \mathcal{A} \rangle, \hat{q})$.

Proof. From Definition 4 it is easily seen that $Var(q) = Var(\hat{q})$ and $q \subseteq \hat{q}$. Hence, $\mathsf{ans}(\langle \mathcal{T}, \mathcal{A} \rangle, \hat{q}) \subseteq \mathsf{ans}(\langle \mathcal{T}, \mathcal{A} \rangle, q)$. For the other direction, let $\mu \in \mathsf{ans}(\langle \mathcal{T}, \mathcal{A} \rangle, q)$. Any function f from Definition 4 is such that $\langle \mathcal{T}, \mathcal{A}_q^f \rangle \models \hat{\mathcal{A}}_q^f$. Hence, $\langle \mathcal{T}, \mathcal{A}_q^\mu \rangle \models \hat{\mathcal{A}}_q^\mu$, i.e., $\langle \mathcal{T}, \mu(q) \rangle \models \mu(\hat{q})$, which means $\langle \mathcal{T}, \mathcal{A} \cup \mu(q) \rangle \models \mu(\hat{q})$ due to monotonicity. But then $\langle \mathcal{T}, \mathcal{A} \rangle \models \mu(\hat{q})$ since $\langle \mathcal{T}, \mathcal{A} \rangle \models \mu(q)$, i.e., $\mu \in \mathsf{ans}(\langle \mathcal{T}, \mathcal{A} \rangle, \hat{q})$. □

4.1 Optimized Query Answering using Query Extension

Since extending the query with all atoms from Definition 4 and evaluating the extended query is not efficient and may result in the checking of many not entailed mappings, we afterwards describe an optimized algorithm that uses only specific extension atoms to reduce the query answering time. A naive algorithm that uses the extension atoms could be as follows: Given a query q and an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, we compute the extended query ABox $\hat{\mathcal{A}}_q^{f,\mathcal{T}}$ and the extended query $\hat{q}$ for some suitable bijection f. Note that we do not have to consider the (often large) ABox $\mathcal{A}$ from $\mathcal{O}$ for computing $\hat{q}$. For each atom $\mathsf{at} \in \hat{q} \setminus q$ we check whether at restricts q according to Definition 3 and create a new query q' that consists of q and the restricting atoms from $\hat{q} \setminus q$. We afterwards evaluate q'.

Although $K[\cdot]$ and $P[\cdot]$ are often fast to compute (due to the use of simpler approximate reasoning algorithms or since the sets are simply extracted from a pre-model) and often cached, it is not very efficient to always retrieve all required such sets from the reasoner in order to perform the required joins for determining the restricting atoms. Hence, in Algorithm 3 an alternative definition for restricting atoms that just uses the cardinalities of the sets $K[\cdot]$ and $P[\cdot]$ is used. The idea is to use the cardinalities of restricting atoms from $\hat{q}$ to more precisely estimate the cardinalities of atoms in q. This allows for a better ordering of query atoms in cost-based query planning. Moreover, since it is expensive to evaluate restricting atoms if they do not contribute relevant mappings, in Algorithm 3 we show how we can use the query extension technique for query optimization while checking only relevant mappings.

Algorithm 3 (evaluateExtensionQans) takes as input a query q and an ontology $\mathcal{O}$ and produces a set of certain answers for q over $\mathcal{O}$. The goal for efficiency is to reduce the sets $K[\cdot]$ and $P[\cdot]$ of query atoms from q based on atoms on the extension of q ($\hat{q}$) created as in Definition 4. Note that we use the methods $\mathsf{inst_K}(\mathcal{O}, \mathsf{at})$ and $\mathsf{inst_P}(\mathcal{O}, \mathsf{at})$ to retrieve the sets $K[\mathsf{at}]$ and $P[\mathsf{at}]$ of at and the methods $\mathsf{inst_{|K|}}(\mathcal{O}, \mathsf{at})$ and $\mathsf{inst_{|P|}}(\mathcal{O}, \mathsf{at})$ to retrieve the cardinalities of the sets $K[\mathsf{at}]$ and $P[\mathsf{at}]$ of at, respectively. In particular, we first take the extended query $\hat{q}$ according to Definition 4 (line 4) and for each atom at_q of q that can be restricted, we check which of the extension atoms (i.e., atoms in $\hat{q} \setminus q$) are restricting for at_q and we keep in $RA[\mathsf{at}_q]$ the restricting atom that leads to the smallest $P[\mathsf{at}_q]$ set for at_q (lines 5-16). The method canbeRestricted takes as input an atom at_q of the initial query and returns true if at_q is of the form $C(x)$, $r(x, a)$ or $r(a, x)$, otherwise it returns false. An atom at_r is considered a restricting atom if it shares variables with at_q and additionally it reduces the $P[\cdot]$ set of at_q, i.e., the cardinality of the $K[\cdot]$ and $P[\cdot]$ sets of at_r is smaller than the cardinality of the $P[\cdot]$ set of at_q (line 11).

After defining restricting atoms for every atom $\mathsf{at}_q \in q$ we move to the evaluation of the query. We first order the atoms in q based on the sets of known and the reduced sets of possible mappings for query atoms, which are created using information about the restricting atoms from the structure RA (line 17). Note that in contrast to Algorithm 1, the method order in Algorithm 3 additionally takes RA into account. We initialize the set of certain answers R_{ans} with the identity mapping μ_0 which does not map any variable to any value (line 18). For each atom at_q of q and mapping $\mu \in R_{ans}$ we proceed as described below (lines 19-33 of Algorithm 3): If at_q instantiated by μ has all its variables bound we check if the appropriate projection of the mapping belongs to the $K[\cdot]$

Algorithm 3 evaluateExtesionQans($\mathcal{O}, q$)

Require: $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$: an OWL 2 DL ontology
q: a query over $\mathcal{O}$
Ensure: the certain answers for q over $\mathcal{O}$

1: **for** $\mathsf{at}_q \in q$ **do**
2: $RA[\mathsf{at}_q] := \mathsf{at}_q$
3: **end for**
4: $\hat{q} := \mathsf{extend}(q, \mathcal{O})$
5: **for** $\mathsf{at}_q \in q$ **do**
6: **if** $\mathsf{canbeRestricted}(\mathsf{at}_q)$ **then**
7: $P_N^{\mathsf{at}_q} = \mathsf{inst}_{|P|}(\mathcal{O}, \mathsf{at}_q)$
8: **for** $\mathsf{at}_r \in \hat{q} \setminus q$ **do**
9: $\langle K_N^{\mathsf{at}_r}, P_N^{\mathsf{at}_r} \rangle = \langle \mathsf{inst}_{|K|}(\mathcal{O}, \mathsf{at}_r), \mathsf{inst}_{|P|}(\mathcal{O}, \mathsf{at}_r) \rangle$
10: **if** $Var(\mathsf{at}_q) \cap Var(\mathsf{at}_r) \neq \emptyset$ and $K_N^{\mathsf{at}_r} + P_N^{\mathsf{at}_r} < P_N^{\mathsf{at}_q}$ **then**
11: $P_N^{\mathsf{at}_q} = K_N^{\mathsf{at}_r} + P_N^{\mathsf{at}_r}$
12: $RA[\mathsf{at}_q] = \mathsf{at}_r$
13: **end if**
14: **end for**
15: **end if**
16: **end for**
17: $\mathsf{at}_1, \ldots, \mathsf{at}_n := \mathsf{order}(q, \mathcal{O}, RA)$
18: $R_{ans} := \{\mu_0 \mid \mathrm{dom}(\mu_0) = \emptyset\}$
19: **for** $i = 1, \ldots, n$ **do**
20: $R := \emptyset$
21: $\mathsf{at}' := RA[\mathsf{at}_i]$
22: **for** $\mu \in R_{ans}$ **do**
23: **if** $Var(\mathsf{at}_i) \setminus \mathrm{dom}(\mu) = \emptyset$ **then**
24: **if** $\mu_{|Var(\mathsf{at}_i)} \in \mathsf{inst}_{\mathsf{K}}(\mathcal{O}, \mathsf{at}_i)$ or $(\mu_{|Var(\mathsf{at}')} \in \mathsf{inst}_{\mathsf{P}}(\mathcal{O}, \mathsf{at}')$ and $\mathcal{O} \models \mu(\mathsf{at}_i))$ **then**
25: $R := R \cup \{\mu\}$
26: **end if**
27: **else**
28: $R := R \cup \{\mu' \cup \mu \mid \mu' \in \mathsf{inst}_{\mathsf{K}}(\mathcal{O}, \mu(\mathsf{at}_i))\}$
29: $R := R \cup \{\mu' \cup \mu \mid \mu' \in \mathsf{inst}_{\mathsf{P}}(\mathcal{O}, \mu(\mathsf{at}')), \mathcal{O} \models (\mu' \cup \mu)(\mathsf{at}_i)\}$
30: **end if**
31: **end for**
32: $R_{ans} := R$
33: **end for**
34: **return** R_{ans}

set of the atom or if it belongs to the $P[\cdot]$ set of the restricting atom and the mapping leads to an entailed axiom (lines 23-26). If at_q contains unbound variables we extend μ with mappings for the unbound variables based on the $K[\cdot]$ set of the atom and the $P[\cdot]$ set of the restricting atom that lead to entailed axioms (lines 27-30). Note that we check if the mapping belongs to the possible set of the restricting atom but we check entailment using at_q of q since we are interested in the evaluation of the atoms of q. Also note that the mappings μ' do not assign values to any of the variables covered by

the already computed (partial) solution μ. This allows for defining the union of μ and μ' by setting $(\mu \cup \mu')(v) = \mu(v)$ if $v \in \text{dom}(\mu)$, and $(\mu \cup \mu')(v) = \mu'(v)$ otherwise.

5 Evaluation

Although the proposed optimized algorithm can be used, in general, for improving the performance of most approximate query answering algorithms, here the evaluation is based on the system described in [5]. The evaluation of the proposed optimized algorithm has been performed using the HermiT reasoner[3] and extending the OWL-BGP system.[4] For extracting the $K[\cdot]$ and $P[\cdot]$ sets for each query atom (which we call known and possible instance sets respectively) we use the pre-model of the queried ontology generated by HermiT to read-off known (possible) instances of classes and properties by analyzing which facts have been added deterministically (non-deterministically) to the pre-model as it is described in our previous work [5]. We do not extract information about the instances of (complex) class expressions from the pre-model, hence we assume that complex class expressions have only possible instances and these are all the individuals appearing in the signature of the queried ontology.

We tested the developed optimizations with the University Ontology Benchmark (UOBM) [6] and a range of custom queries that show the effect of the presented techniques. All experiments were performed on a Mac OS X Lion machine with a 2.53 GHz Intel Core i7 processor and Java 1.6 allowing 1GB of Java heap space. Note that UOBM contains disjunctions and the reasoner makes also nondeterministic derivations. In order to reduce the reasoning time, we removed the nominals which are hard to deal with and we used the first department of UOBM containing 3,043 individuals and 15,250 ABox assertions. The resulting ontology took 16 s to load and 0.1 s to classify and initialize the known and possible instances.

In order to show the effect that query extension (Algorithm 3) has, we have created the following queries, which contain atoms with possible instances:

$$\mathsf{q}_1 = \{\mathsf{isAdvisedBy}(x, y), \mathsf{GraduateStudent}(x), \mathsf{Woman}(y)\}$$
$$\mathsf{q}_2 = \{\mathsf{isTaughtBy}(x, y), \mathsf{GraduateCourse}(x), \mathsf{Woman}(y)\}$$
$$\mathsf{q}_3 = \{\mathsf{teachingAssistantOf}(x, y), \mathsf{GraduateCourse}(y), \mathsf{Woman}(x)\}$$
$$\mathsf{q}_4 = \{\exists \mathsf{takesCourse}.\top(x), \forall \mathsf{takesCourse}.\mathsf{GraduateCourse}(x)\}$$
$$\mathsf{q}_5 = \{\mathsf{Woman}(x), \exists \mathsf{worksFor}.\mathsf{Organization}(x)\}$$

In the above queries, the sets $P[\mathsf{GraduateStudent}]$, $P[\mathsf{Woman}]$ and $P[\mathsf{GraduateCourse}]$ are non-empty, i.e., the query classes have possible instances. In Table 1 we compare the number of possible instances that are being checked (i.e., the number of performed consistency checks), denoted by EntNo in the table, and the running time of three different algorithms w.r.t. the above queries, i.e., i) Algorithm evaluateExtensionQans, ii) the algorithm from our previous work when static ordering is used [5], which is denoted by Static in the table and iii) Algorithm evaluateIntersecQans. Algorithm Static takes as input a query and an ontology and orders the atoms of the query based on the known and possible instances of query classes and properties. Then, the query evaluation starts with the first atom, retrieves the known mappings and checks all remaining

[3] http://hermit-reasoner.com/
[4] https://code.google.com/p/owl-bgp/

Table 1. Query answering times in seconds for UOBM with one dep using i) the static algorithm from [5], ii) evaluateIntersecQans and iii) evaluateExtensionQans

	Static		evaluateIntersecQans		evaluateExtensionQans	
Query	Time	EntNo	Time	EntNo	Time	EntNo
q1	20.84	47	11.79	29	10.54	19
q2	21.63	51	10.70	26	12.06	26
q3	12.78	32	5.05	12	5.60	12
q4	569.06	1694	1,216.18	3709	256.77	1332
q5	2,198.66	2117	829.64	2198	18.36	135

possible mappings using either dedicated reasoner methods or entailment checks. While the former is quite cheap, involving only look-ups to the memory, the latter is usually significantly more expensive, involving reasoning procedures. For the next query atom, the evaluation is analogous with the difference that the join variables are taken into account, i.e., the evaluation is restricted only to already established mappings if a joined variable has been evaluated in a previous step.

From Table 1 we see that Algorithm evaluateExtensionQans considerably reduces the number of performed consistency checks in comparison to Static and hence the query answering times for all queries. As explained in the previous section this is achieved by exploiting the known and possible instances of atoms belonging to the extension of the query. For example, in Query q_1 Algorithm Static results in the ordering [isAdvisedBy(x,y), GraduateStudent(x), Woman(y)], while the use of the extension atom Professor(y) significantly reduces the query specific instances of Woman(y) and hence the ordering [isAdvisedBy(x, y), Woman(y), GraduateStudent(x)] is chosen based on the reduced instance sets, which leads to better performance results. Similarly, for Queries q_2 and q_3 the use of the extension atoms Facutly(y) and TeachingAssistant(x) significantly reduces the mappings for the atoms Woman(y) and Woman(x) respectively. In the same way for Queries q_4 and q_5, which contain atoms with (complex) class expressions, the use of the extension atoms GraduateStudent(x) and Employee(x) significantly reduces the number of mappings for x that need to be checked. Note that the query extension phase takes around 0.3 seconds for all the above queries. Algorithm evaluateIntersecQans also performs better than Static because only the possible mappings that belong to the known or possible sets of all query atoms are checked.

Regarding the comparison between the algorithms evaluateExtensionQans and evaluateIntersecQans we see that evaluateExtensionQans performs much better than evaluateIntersecQans for queries containing atoms with (complex) class expressions and it performs similarly for queries without atoms with (complex) class expressions. It is worth noting that on Query q_5 Static performs worse than evaluateIntersecQans even though Static performs less consistency checks than evaluateIntersecQans. This happens because the two algorithms choose a different ordering and, in particular, Static evaluates the atom $\exists$worksFor.Organization(x) first, which requires more expensive consistency checks than the complexity of the consistency checks performed for the atom Woman(x) and the afterwards evaluation of $\exists$worksFor.Organization(x) based on the reduced x-mappings.

6 Conclusions

In the current paper we presented an approach for using the TBox of an ontology to optimize the evaluation of SPARQL instance queries evaluated under the OWL Direct Semantics entailment regime. For those queries we showed how we can build equivalent queries with additional atoms which can be exploited to reduce the set of possible mappings for query variables. Through our experimental evaluation we showed that the use of these extension atoms can lead to a significant reduction in query answering time, which can be up to two orders of magnitude.

Acknowledgements This work was supported by IKY in collaboration with DAAD in the program IKYDA: Automatic data generation for description logic knowledge bases.

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, second edn. (2007)
2. Calvanese, D., De Giacomo, G., Lenzerini, M.: Conjunctive query containment and answering under description logics constraints. ACM Transactions on Computational Logic 9(3) (2008)
3. Glimm, B., Kazakov, Y., Kollia, I., Stamou, G.: Using the TBox to optimise SPARQL queries. In: Proceedings of the 2013 International Description Logic Workshop (DL 2013). CEUR Workshop Proceedings (2013)
4. Glimm, B., Ogbuji, C.: SPARQL 1.1 entailment regimes. W3C Recommendation (21 March 2013), available at http://www.w3.org/TR/sparql11-entailment/
5. Kollia, I., Glimm, B.: Optimizing SPARQL Query Answering over OWL Ontologies. J. Artif. Intell. Res. (JAIR) 48, 253–303 (2013)
6. Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y., Liu, S.: Towards a complete OWL ontology benchmark. In: The Semantic Web: Research and Applications, pp. 125–139. Lecture Notes in Computer Science, Springer (2006)
7. Pan, J.Z., Thomas, E.: Approximating OWL-DL Ontologies. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence. pp. 1434–1439. AAAI Press (2007)
8. Pan, J.Z., Thomas, E., Zhao, Y.: Completeness Guaranteed Approximation for OWL DL Query Answering. In: Proceedings of the 2009 International Workshop on Description Logics (DL'09) (2009)
9. Patel-Schneider, P.F., Motik, B. (eds.): OWL 2 Web Ontology Language: Mapping to RDF Graphs. W3C Recommendation (27 October 2009), available at http://www.w3.org/TR/owl2-mapping-to-rdf/
10. Prud'hommeaux, E., Seaborne, A. (eds.): SPARQL Query Language for RDF. W3C Recommendation (15 January 2008), available at http://www.w3.org/TR/rdf-sparql-query/
11. Ren, Y., Pan, J.Z., Zhao, Y.: Soundness Preserving Approximation for TBox Reasoning. In: Proceedings of the 25th National Conference on Artificial Intelligence (AAAI'10). AAAI Press (2010)
12. Zhou, Y., Nenov, Y., Grau, B.C., Horrocks, I.: Complete query answering over horn ontologies using a triple store. In: Proceedings of the 12th International Semantic Web Conference (ISWC'13). Lecture Notes in Computer Science, vol. 8218, pp. 720–736. Springer-Verlag (2013)