Using the TBox to Optimise SPARQL Queries

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. We present an approach for using schema knowledge from the TBox to optimise the evaluation of SPARQL queries. The queries are evaluated over an OWL ontology using the OWL Direct Semantics entailment regime. For conjunctive instance queries, we proceed by transforming the query into an 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. We also consider arbitrary SPARQL queries and show how the concept and role hierarchies can be used to prune the search space of possible answers based on the polarity of variable occurrences in the query.

1 Introduction

In this paper, we consider the SPARQL 1.1 query language [7], which was recently standardised by the World Wide Web Consortium (W3C). SPARQL 1.1 includes several *entailment regimes* [4] in order to use entailment when evaluating a query. In this paper, we consider SPARQL queries evaluated using OWL's Direct Semantics Entailment [14]. In this setting, the WHERE clause or query pattern of a query can be seen as a set of extended OWL or Description Logic (DL) axioms, which can have variables in place of concept, role or individual names. Our goal in this paper is to optimise the evaluation of such query patterns.

Over the last decade, much effort has been spent on optimising standard reasoning tasks such as entailment checking, classification, or realisation (i.e., the computation of instances of all concepts and roles) [3, 18, 22]. The optimisation of query answering algorithms has, however, mostly been addressed for conjunctive queries in OWL profiles, most notably the OWL 2 QL profile [2, 13, 16]. An exception to this are the works on nRQL [6], SPARQL-DL [19], and our previous work [12].[3] The query language nRQL is supported by Racer Pro [5] and allows for queries that go beyond ABox queries, e.g., one can retrieve sub- or super-concepts of a given concept. SPARQL-DL is a fragment of SPARQL implemented in the Pellet reasoner [20]. The kinds of SPARQL queries that are supported in SPARQL-DL are those that can directly be mapped to reasoner tasks. Furthermore, KAON2 [9] supports SPARQL queries, but restricted to ABox queries/conjunctive instance queries. In our previous work [12], we propose algorithms for arbitrary SPARQL queries and optimisation techniques mainly based on cost-based query planning. The system is implemented as a SPARQL Wrapper that can

[3] An implementation is available at http://code.google.com/p/owl-bgp/

be used with any reasoner that implements the OWLReasoner interface of the OWL API [8]. Finally, TrOWL[4] supports SPARQL queries based on our SPARQL wrapper, but the reasoning in TrOWL is approximate, i.e., an OWL DL ontology is rewritten into an ontology that uses a less expressive language before reasoning is applied [21]. A promising, but still very preliminary work is also the Semantic Web Entailment Regime Translation and Inference Architecture (Swertia);[5] a generic Semantic Web reasoning framework that is based on first-order logic (FOL) reasoning.

The contribution of this paper is two-fold: first, we present an optimisation that is applicable to conjunctive instance queries. We show that one can compute an equivalent query $\hat{q}$ for a given query q by replacing the variables in q with fresh individual names. We then perform realisation, i.e., we materialise entailed concept and role 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. Second, we propose an optimisation for also reducing the possible mappings for concept and role variables by exploiting the polarity of variable occurrences in the query and the (precomputed) concept and role hierarchies. We provide a prototypical implementation and evaluation of the polarity based optimisation, which shows that it can lead to an improvement of up to two orders of magnitude in the execution times of some queries. The polarity based optimisation can be found in more detail in our earlier work [11].

2 Preliminaries

Due to lack of space, we do not introduce the Description Logic $\mathcal{SHIQ}$, which we use throughout the paper. For further details, we refer interested readers to the DL handbook [1]. We assume that a *knowledge base* $\mathcal{K}$ is a pair $(\mathcal{T}, \mathcal{A})$ with $\mathcal{T}$ a TBox that possibly includes role inclusion axioms and $\mathcal{A}$ an ABox.

In this paper, we consider conjunctive instance queries and complex queries, which can also contain axioms with variables in place of concept or role names. Such queries are important in the context of SPARQL since SPARQL's OWL Direct Semantics entailment regime allows for such queries. Conjunctive instance queries are a subset of all such SPARQL queries, but they differ from database-style conjunctive queries in that they do not allow for existentially quantified or non-distinguished variables.

Definition 1 (Conjunctive Instance and Complex Queries). *Let $\mathcal{S} = (N_C, N_R, N_I)$ be a signature, $\mathcal{K}$ a knowledge base over $\mathcal{S}$, and $V = V_C \uplus V_R \uplus V_I$ a countably infinite set of* variable names *(*concept variables *in V_C,* role variables *in V_R, and* individual variables *in V_I) disjoint from N_C, N_R, and N_I. Let $A \in N_C$, $r \in N_R$, and $x, y \in V_I$. A* concept atom *is an expression $A(x)$ and a* role atom *is an expression $r(x, y)$. A* conjunctive instance query *q is a non-empty set of (concept or role) atoms. The set of $\mathcal{SHIQ}$* concept templates *(or concept templates for short) over $\mathcal{S}$ and V is built as the set of $\mathcal{SHIQ}$ concepts, where a concept variable can be used in place of a concept name, and a role variable in place of a role name. A* role axiom template *has the form $r \sqsubseteq s$ where $r, s \in N_R \cup V_R$.*

[4] http://trowl.eu
[5] http://swertia.org

A concept axiom template *has the form $c \sqsubseteq d$ with c, d concept templates. We again abbreviate $c \sqsubseteq d$ and $d \sqsubseteq c$ as $c \equiv d$. A finite set of role axiom templates, concept axiom templates, and (concept or role) atoms is called a* complex query. *We use $Var(q)$ for the set of variables in q. and $|Var(q)|$ is called the* arity *of q.*

Let $q = \{\mathsf{at}_1, \ldots, \mathsf{at}_n\}$ be a (conjunctive instance or complex) query. A total function $\mu\colon Var(q) \to N_C \cup N_R \cup N_I$ is a mapping for q over $\mathcal{K}$ *if $\mu(v) \in N_C$ if $v \in V_C$, $\mu(v) \in N_R$ if $v \in V_R$, and $\mu(v) \in N_I$ if $v \in V_I$. Let $X = \{x_1, \ldots, x_n\} \subseteq Var(q)$ a subset of the variables of q, and $M = \{\mu_1, \ldots, \mu_m\}$ a set of mappings for q. The* projection of X over M *is the set $M_{|X} = \{\{x \mapsto a\} \mid \mu \in M, x \in X \text{ and } \mu(x) = a\}$. A mapping μ is a* certain answer for q over $\mathcal{K}$ *if $\mathcal{K} \models \mu(\mathsf{at})$ for each $\mathsf{at} \in q$, written $\mathcal{K} \models \mu(q)$, where $\mu(\mathsf{at})$ ($\mu(q)$) is the result of replacing each $v \in Var(\mathsf{at})$ ($Var(q)$) with $\mu(v)$. We denote the set of all certain answers for q over $\mathcal{K}$ with* $\mathsf{ans}(\mathcal{K}, q)$.

3 Query Answering via Approximate Instance Retrieval

In this section, we use the DL $\mathcal{SHIQ}$ and we present a technique that uses approximate reasoning algorithms in order to optimise the evaluation of conjunctive instance queries. Approximate reasoning algorithms can either be sound and incomplete [17], i.e., they underapproximate the set of certain or known answers or they can be complete but unsound [15], i.e., they overapproximate 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 overapproximation. Another possibility is to use a pre-model or complete and clash free tableau generated by a DL reasoner [12]. One can then read-off certain instances of concepts or roles by analysing which concept and role facts have been added deterministically to the pre-model, i.e., one can obtain an underapproximation for concept and role instances. Similarly, one can analyse the non-deterministically added and absent concept and role facts to compute an overapproximation. More formally, we define an approximate instance retrieval algorithm as follows:

Definition 2 (Approximate Instance Retrieval Algorithm). *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a knowledge base and at a concept or role atom such that the concept or role is from the signature of $\mathcal{K}$. An* approximate instance retrieval algorithm $\mathsf{inst}(\mathcal{K}, \mathsf{at})$ *returns a pair of sets of (total) functions from $Var(\mathsf{at})$ to individual names from $\mathcal{K}$, $\langle K[\mathsf{at}], P[\mathsf{at}] \rangle$, such that:*

1. *if $\mu \in K[\mathsf{at}]$, then $\mathcal{K} \models \mu(\mathsf{at})$, and*
2. *for each total function μ from $Var(\mathsf{at})$ to individual names from $\mathcal{K}$ such that $\mathcal{K} \models \mu(\mathsf{at})$, $\mu \in K[\mathsf{at}] \cup P[\mathsf{at}]$.*

Similarly, we define approximate query answering algorithms:

Definition 3 (Approximate Query Answering Algorithm). *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a knowledge base, and q a conjunctive instance query. An* approximate query answering algorithm $\mathsf{apprQA}(\mathcal{K}, q)$ *returns a pair of sets of (total) functions from $Var(q)$ to individual names from $\mathcal{K}$ $\langle K[q], P[q] \rangle$ such that:*

1. *if* $\mu \in K[q]$*, then* $\mu \in \mathsf{ans}(\mathcal{K}, q)$*, and*
2. *for each total function* μ *from* $Var(q)$ *to individual names from* $\mathcal{K}$ *such that* $\mu \in \mathsf{ans}(\mathcal{K}, q)$*,* $\mu \in K[q] \cup P[q]$*.*

Without loss of generality, in the rest of the paper we assume that for every approximate instance retrieval or query answering algorithm $K[\cdot] \cap P[\cdot] = \emptyset$ holds. It is not difficult to see that an obvious approach to develop an approximate query answering algorithm apprQA is to use an approximate instance retrieval algorithm inst. A naive method to do this is to execute inst and take $K[\cdot] \cup P[\cdot]$ for each query atom and then compute the join of these sets, where we straightforwardly interpret the mappings as relations. The following example illustrates that some possible answers can be easily rejected.

For ease of presentation we represent mappings as tuples in the examples that follow, i.e., abusing notation, $K[\mathsf{at}]$ is a set of individuals for concept atoms and a set of pairs of individuals for role atoms and $K[q]$ is a set of n-tuples, n being the arity of q.

Example 1. Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a knowledge base and $q = \{C(x), r(x, y), D(y)\}$ a query with variables $\langle x, y \rangle$. Suppose that (possibly as a result of $\mathsf{inst}(\mathcal{K}, C(x))$, $\mathsf{inst}(\mathcal{K}, r(x, y))$ and $\mathsf{inst}(\mathcal{K}, D(y))$) we have $K[C(x)] = \{a\}$, $P[C(x)] = \{b\}$, $K[r(x, y)] = \{\langle a, c \rangle\}$, $P[r(x, y)] = \{\langle b, d \rangle, \langle b, e \rangle\}$, $K[D(y)] = \{c\}$ and $P[D(y)] = \{d\}$. Even if we do not know $\mathcal{K}$, we can conclude that $\langle a, c \rangle$ is a certain answer to q, since $a \in K[C(x)]$, $\langle a, c \rangle \in K[r(x, y)]$ and $c \in K[D(y)]$. However, only $\langle b, d \rangle$ is a possible answer for q since $b \in P[C(x)]$, $\langle b, d \rangle \in P[r(x, y)]$ and $d \in P[D(y)]$; $\langle b, e \rangle$ cannot be an answer for q since although $\langle b, e \rangle \in P[r(x, y)]$ and $b \in P[C(x)]$, $e \notin K[D(y)] \cup P[D(y)]$.

Algorithm intersecQans (see Algorithm 1) formalises this idea, which we also illustrate in Example 2:

Example 2. If we take q from Example 1 in the order given, we first initialise $K[q]$ to $\{a\}$ and $P[q]$ to $\{b\}$. During the next iterations, the (preliminary) set $K[q]$ is 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 Example 1, 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.*

Concerning the optimisation of algorithm intersecQans, in practice, one would use well-known ordering strategies from the area of databases for the atoms in q in order to reduce the number of intermediate results, e.g., one would prefer joins over connected atoms and join a small relation with a bigger one where possible. The question now is: given a knowledge base $\mathcal{K}$ and a query q, how we can further reduce the cardinality of $P[q]$ computed by intersecQans with the aid of inst.

Example 3. Suppose that we have $\mathcal{K}$ and $q = \{C(x), r(x, y), D(y)\}$ as in Example 1 and, in addition, we are given that $\mathcal{K} \models \exists r.\top \sqcap C \sqsubseteq B$ and $\mathsf{inst}(\mathcal{K}, B(x)) = \langle \{a\}, \emptyset \rangle$. From Example 2, we have $K[q] = \{\langle a, c \rangle\}$ and $P[q] = \{\langle b, d \rangle\}$. In this case, $\langle b, d \rangle$ is no longer

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

Require: $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$: a $\mathcal{SHIQ}$ knowledge base
q: a conjunctive query
Ensure: $\langle K[q], P[q] \rangle$: $K[q], P[q]$ sets of known and possible answers for q

```
for at ∈ q do
    ⟨K[at], P[at]⟩ := inst(K, at)
    if K[q] and P[q] not initialised then
        ⟨K[q], P[q]⟩ := ⟨K[at], P[at]⟩
    else
        K[q] := K[q] ⋈ K[at]
        P[q] := (P[q] ⋈ P[at]) ∪ (K[q] ⋈ P[at]) ∪ (P[q] ⋈ K[at])
    end if
end for
return ⟨K[q], P[q]⟩
```

a possible answer of q. If b would be a possible mapping for x, it would have an r-successor (since $r(x, y) \in q$) and it would be an instance of C (since $C(x) \in q$) and, hence, b should be in $K[B(x)] \cup P[B(x)]$ to satisfy the entailed axiom.

We now define the notion of restricting atoms such as $B(x)$.

Definition 4 (Restricting Atoms). *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a knowledge base, q a conjunctive instance 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 $\langle K[q], P[q] \rangle = \mathsf{intersecQans}(\mathcal{K}, q)$ and $\langle K[\mathsf{at}], P[\mathsf{at}] \rangle = \mathsf{inst}(\mathcal{K}, \mathsf{at})$.

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

If we want to preserve the certain answers of q, we should only use restricting atoms that do not change the answers of q. Let q and q' be conjunctive instance queries such that $q' = q \cup \{\mathsf{at}\}$. 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 $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a knowledge base, q and q' two queries such that $q' = q \cup \{\mathsf{at}\}$, $\mathsf{ans}(\mathcal{K}, q) = \mathsf{ans}(\mathcal{K}, q')$, $\langle K[q], P[q] \rangle = \mathsf{intersecQans}(\mathcal{K}, q)$, $\langle K[\mathsf{at}], P[\mathsf{at}] \rangle = \mathsf{inst}(\mathcal{K}, \mathsf{at})$, and* at *restricts $P[q]$.*

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

2. $|\{\mu \in P[q] \mid \mu_{|Var(\mathsf{at})} \in (K[\mathsf{at}] \cup P[\mathsf{at}])\}| < |P[q]|$.

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. The proposed query extension technique is analogous to the use of the chase technique in relational database theory to reason about conjunctive query containment [10].

Definition 5 (Query Extension). *Let $\mathcal{T}$ be a TBox, q a query, and f a total and bijective function from $Var(q)$ to a set of individual names from N_I. The* canonical ABox *for q w.r.t. f is defined as follows:*

$$\mathcal{A}_q^f = \{B(f(x)) \mid B(x) \in q\} \cup \{r(f(x), f(y)) \mid r(x, y) \in q\}$$

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

$$\hat{\mathcal{A}}_q^f = \{B(a) \mid B \in N_C, a \in Ind(\mathcal{A}_q^f) \text{ and } \langle \mathcal{T}, \mathcal{A}_q^f \rangle \models B(a)\} \cup$$
$$\{r(a, b) \mid r \in N_R, a, b \in Ind(\mathcal{A}_q^f) \text{ and } \langle \mathcal{T}, \mathcal{A}_q^f \rangle \models r(a, b)\}.$$

The extended query $\hat{q}$ *w.r.t. f and $\hat{\mathcal{A}}_q^f$ is:*

$$\hat{q} = \{B(f^-(a)) \mid B(a) \in \hat{\mathcal{A}}_q^f\} \cup \{r(f^-(a), f^-(b)) \mid r(a, b) \in \hat{\mathcal{A}}_q^f\}.$$

Example 5. Suppose we have $\mathcal{K}$ such that $\mathcal{K} \models \exists r.\top \sqcap C \sqsubseteq B$ and $q = \{C(x), r(x, y), D(y)\}$ from Example 3. In this case, the canonical ABox of q is $\mathcal{A}_q^f = \{C(a_x), r(a_x, a_y), D(a_y)\}$ where f maps a variable $v \in \{x, y, z\}$ to the individual name a_v. The extended canonical ABox for the query q w.r.t. f is $\hat{\mathcal{A}}_q^f = \{C(a_x), r(a_x, a_y), D(a_y), B(a_x)\}$ and the extended query w.r.t. $\hat{\mathcal{A}}_q^f$ is $\hat{q} = \{C(x), r(x, y), D(y), B(x)\}$. Please note that $\hat{q}$ has the same answers as q (w.r.t. any ABox), because $\mathcal{K} \models \exists r.\top \sqcap C \sqsubseteq B$.

Lemma 3. *The extended query $\hat{q}$ created as in Definition 5 is unique.*

Intuitively (see Example 5), the extended query $\hat{q}$ w.r.t. $\hat{\mathcal{A}}_q^f$ adds atoms to q that do not change the set of answers of q, which we formalise by the following Theorem:

Theorem 1. *Let $\mathcal{T}$ be a TBox, q a query, and $\hat{q}$ the extended query as in Definition 5. Then, $\mathcal{T} \models q \equiv \hat{q}$, i.e., for any ABox $\mathcal{A}'$,* $\mathsf{ans}(\langle \mathcal{T}, \mathcal{A}' \rangle, q) = \mathsf{ans}(\langle \mathcal{T}, \mathcal{A}' \rangle, \hat{q})$

Given a conjunctive instance query q and a knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, we can now compute the known and possible answers with Algorithm 1. We then compute the extended canonical ABox $\hat{\mathcal{A}}_q^f$ 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 $\mathcal{K}$ for computing $\hat{q}$. For each atom $\mathsf{at} \in \hat{q} \setminus q$, we can then use our approximate instance retrieval algorithm to check whether at restricts q and reduce the set of possible answers $P[q]$ accordingly.

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. Hence, in our future work, we plan to just use the cardinalities of the sets $K[\cdot]$ and $P[\cdot]$. The idea is to use the cardinalities of restricting atoms from $\hat{q}$ to more precisely estimate the cardinalities of atoms in q. This will allow for a better ordering of the query atoms in cost-based query planning.

4 Beyond Conjunctive Instance Queries

Since conjunctive instance queries are only a subset of the queries that are important in the context of SPARQL, we present, in this section, an optimisation that aims at improving the performance for evaluating arbitrary complex queries. In general for a complex query $q = \{\mathsf{at}_1, \ldots, \mathsf{at}_n\}$, we partition q into two sets q_s and q_c such that q_s is the maximal conjunctive instance sub-query of q and q_c contains the axiom templates. We first evaluate q_s, e.g., by employing optimisation techniques as described in the previous section. The intermediate results can then already be used to reduce the possible answers for q_c. In order to optimise the evaluation of q_c, we assume that the concepts and roles are classified (this is often done before a system accepts any queries), which means that axiom templates of the form $x \sqsubseteq A$ with $x \in V_C$, $A \in N_C$ require only cache lookups. We use the hierarchies to prune the search space of solutions in the evaluation of certain axiom templates as illustrated with the following example:

Example 6. Let $q = \{\text{Infection} \sqsubseteq \exists \text{hasCausalLinkTo}.x\}$ with $x \in V_C$ a concept variable and $\mathcal{K}$ some knowledge base. If, mapping x to the concept name A, does not yield an entailed axiom and $\mathcal{K} \models B \sqsubseteq A$ holds, then B is also not a certain answer. Thus, when searching for certain answers, we choose the next binding to test by traversing the concept hierarchy top-down. When we find a non-answer A, the subtree rooted in A of the concept hierarchy can safely be pruned. Queries over ontologies with a large number of concepts and a deep concept hierarchy can, therefore, gain the maximum advantage from this optimisation. We employ similar optimisations using the role hierarchy.

In the example above, we can prune the subconcepts of A because x has positive polarity in the axiom, i.e., x occurs positively and not under a negation. In case a variable occurs negatively, e.g., directly or indirectly under a negation or positively on the left-hand side of an axiom, one can, instead, prune the superclasses. We next specify more precisely the polarity of a concept variable in a concept axiom template.

Definition 6. *Let $x \in V_C$ be a concept variable and C, C_1, C_2, D concept axiom templates, r a role, and $n \in \mathbf{N}_0$. We define the* polarity *of x in D as follows: x* occurs positively *in x. Furthermore, x* occurs positively (negatively) in

- *$\neg C$ if x occurs negatively (positively) in C,*
- *$C_1 \sqcap C_2$ or $C_1 \sqcup C_2$ if x occurs positively (negatively) in C_1 or C_2,*
- *$\exists r.C$, $\forall r.C$, or $\geqslant n\ r.C$ if x occurs positively (negatively) in C,*
- *$\leqslant n\ r.C$ if x occurs negatively (positively) in C*
- *$= n\ r.C$ if x occurs in C.*

We further say that x occurs positively (negatively) in $C_1 \sqsubseteq C_2$ if x occurs negatively (positively) in C_1 or positively (negatively) in C_2. We further define a partial function pol_c that maps a concept variable x and a concept template C (axiom template of the form $C_1 \sqsubseteq C_2$) to pos *if x occurs* only positively *in C ($C_1 \sqsubseteq C_2$) and to* neg *if x occurs* only negatively *in C ($C_1 \sqsubseteq C_2$).*

Note that x can also occur both positively and negatively. Note also that no matter whether x occurs positively or negatively in a concept C, in any concept of the form

$(= n\ r.C)$, x occurs positively as well as negatively. This is due to the fact that $(= n\ r.C)$ is equivalent to the concept template $\leqslant n\ r.C \sqcap \geqslant n\ r.C$ in which x occurs positively as well as negatively in C.

Since the function pol_c is not defined for variables that appear both positively and negatively, the concept hierarchy cannot be exploited in this case. For example, consider the concept axiom template $\neg x \sqcup \exists r.x$ with $x \in V_C$ (i.e., $x \sqsubseteq \exists r.x$), where x appears negatively in $\neg x$ and positively in $\exists r.x$. Now, let $\delta \in \Delta^{\mathcal{I}}$ be an arbitrary element from a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of the queried knowledge base. It is obvious that if δ is an instance of $\neg A \sqcup \exists r.A$ and either $A \sqsubseteq B$ or $B \sqsubseteq A$ holds, we cannot deduce that δ is an instance of $\neg B \sqcup \exists r.B$.

The following theorem holds for every axiom template of the form $C_1 \sqsubseteq C_2$. We use $C_{\mu(x)=A}$ with $A \in N_C$ to denote the concept obtained by applying the extension of μ that also maps x to A.

Theorem 2. *Let $\mathcal{K}$ be a knowledge base, A, B concept names such that $\mathcal{K} \models A \sqsubseteq B$, C_1, C_2 concept templates, $C = \neg C_1 \sqcup C_2$, $x \in V_C$ a concept variable occurring in C, and μ a mapping that covers all variables of C apart from x.*

1. *If $\mathsf{pol}_c(x, C) = \mathsf{pos}$ and $\mathcal{K} \not\models (C_1 \sqsubseteq C_2)_{\mu(x)=B}$, then $\mathcal{K} \not\models (C_1 \sqsubseteq C_2)_{\mu(x)=A}$.*
2. *If $\mathsf{pol}_c(x, C) = \mathsf{neg}$ and $\mathcal{K} \not\models (C_1 \sqsubseteq C_2)_{\mu(x)=A}$, then $\mathcal{K} \not\models (C_1 \sqsubseteq C_2)_{\mu(x)=B}$.*

We now extend this optimisation to the case of role variables and we first define the polarity of a role variable in a concept axiom template.

Definition 7. *Let $x \in V_R$ be a role variable, C, C_1, C_2, D concept templates, r a role, and $n \in \mathbf{N}_0$. We define the* polarity *of x in D as follows: x* occurs positively *in $\exists x.C$, $\exists x^-.C$, $\geqslant n\ x.C$, $\geqslant n\ x^-.C$, $= n\ x.C$, and $= n\ x^-.C$; x* occurs negatively *in $\forall x.C$, $\forall x^-.C$, $\leqslant n\ x.C$, $\leqslant n\ x^-.C$, $= n\ x.C$, and $= n\ x^-.C$. Furthermore, x occurs positively (negatively) in*

- *$\neg C$ if x occurs negatively (positively) in C,*
- *$C_1 \sqcap C_2$ or $C_1 \sqcup C_2$ if x occurs positively (negatively) in C_1 or C_2,*
- *$\exists r.C$, $\exists x.C$, $\exists x^-.C$, $\geqslant n\ r.C$, $\geqslant n\ x.C$, $\geqslant n\ x^-.C$, $\forall r.C$, $\forall x.C$, or $\forall x^-.C$ if x occurs positively (negatively) in C,*
- *$\leqslant n\ r.C$, $\leqslant n\ x.C$, or $\leqslant n\ x^-.C$ if x occurs negatively (positively) in C,*
- *$= n\ r.C$ if x occurs in C.*

We further say that x occurs positively (negatively) in $C_1 \sqsubseteq C_2$ if x occurs negatively (positively) in C_1 or positively (negatively) in C_2. We define a partial function pol_r that maps a role variable x and a concept template C (axiom template of the form $C_1 \sqsubseteq C_2$) to pos *if x occurs* only positively *in C ($C_1 \sqsubseteq C_2$) and to* neg *if x occurs* only negatively *in C ($C_1 \sqsubseteq C_2$).*

We now show, that the hierarchy optimisation is also applicable to role variables, provided they occur only positively or only negatively.

Theorem 3. *Let $\mathcal{K}$ be a knowledge base, r, s role names such that $\mathcal{K} \models r \sqsubseteq s$, C_1, C_2 concept templates, $C = \neg C_1 \sqcup C_2$, $x \in V_R$ a role variable occurring in C, and μ a mapping that covers all variables of C apart from x.*

Algorithm 2 getPossibleConceptVarMappings($x, \mu, \mathsf{at}, \mathcal{K}$)

Require: x: a concept variable, μ: a partial mapping with $x \in \mathrm{dom}(\mu)$
at: an axiom template in which x occurs, $\mathcal{K}$: a $\mathcal{SHIQ}$ knowledge base
Ensure: a set of possible answers for x
if $\mathsf{pol}_c(x, \mathsf{at}) == \mathsf{pos}$ **then**
return $\{\mu' \mid \mu'(x) = C, C$ is direct subconcept of $\mu(x)$ in $\mathcal{K}, \mu'(y) = \mu(y)$ for $y \in \mathrm{dom}(\mu) \setminus \{x\}\}$
else
return $\{\mu' \mid \mu'(x) = C, C$ is direct superconcept of $\mu(x)$ in $\mathcal{K}, \mu'(y) = \mu(y)$ for $y \in \mathrm{dom}(\mu) \setminus \{x\}\}$
end if

1. *If* $\mathsf{pol}_r(x, C) = \mathsf{pos}$ *and* $\mathcal{K} \not\models (C_1 \sqsubseteq C_2)_{\mu(x)=s}$, *then* $\mathcal{K} \not\models (C_1 \sqsubseteq C_2)_{\mu(x)=r}$.
2. *If* $\mathsf{pol}_r(x, C) = \mathsf{neg}$ *and* $\mathcal{K} \not\models (C_1 \sqsubseteq C_2)_{\mu(x)=r}$, *then* $\mathcal{K} \not\models (C_1 \sqsubseteq C_2)_{\mu(x)=s}$.

Algorithm 2 shows how we use the above theorems to create possible concept mappings for a concept variable x that appears only positively or only negatively in an axiom template $C_1 \sqsubseteq C_2$. We can define a similar algorithm for role variables. Hence, an algorithm for evaluating complex queries can call these algorithms to prune the search space once a solution for such a concept or role variable has been found, e.g., by checking entailment for an instantiated axiom template.

4.1 Complex Axiom Template Evaluation

In the absence of suitable standard benchmarks for complex queries or SPARQL queries over OWL ontologies, we created a custom set of queries as shown in Tables 1 and 2 for the GALEN and the FBbt_XP ontology, respectively. Systems that fully support the SPARQL Direct Semantics entailment regime are still under development, which makes it hard to compare our results for these kinds of queries with other systems. 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. We measure the time for one-off tasks such as classification separately since such tasks are usually performed before the system accepts queries. The ontologies and all code required to perform the experiments are available online.[6] For the evaluation we have used the HermiT[7] hypertableau reasoner.

GALEN is a biomedical knowledge base. Its expressivity is (Horn-)$\mathcal{SHIF}$ and it consists of 2,748 concepts and 413 abstract roles. FBbt_XP is an ontology taken from the Open Biological Ontologies (OBO) Foundry.[8] Its expressivity is $\mathcal{SHI}$ and the knowledge base consists of 7,221 concepts and 21 roles. We only consider the TBox part of FBbt_XP since the ABox is not relevant for showing the effects of the optimisation for concept axiom templates. GALEN took 3.7 s to load and 11.1 s to classify (concepts and roles), while FBbt_XP took 1.5 s to load and 7.4 s to classify.

For each query, we tested the execution once with (bold results) and once without the proposed optimisation. The tables also show the number of consistency checks that were performed for the evaluation of each query.

[6] http://code.google.com/p/query-ordering/
[7] http://www.hermit-reasoner.com/
[8] http://www.obofoundry.org/

Table 1. Queries (with $x_{(i)}$ a concept and $y_{(i)}$ a role variable) and evaluation times in seconds for the GALEN knowledge base with (bold) and without the hierarchy exploitation

Query	Consistency Checks	Time in sec.	Answers
Infection $\sqsubseteq \exists$hasCausalLinkTo.x	2,750 **50**	1.68 **0.18**	10
Infection $\sqsubseteq \exists y.x$	1,141,250 **1,291**	578.98 **10.00**	214
$x_1 \sqsubseteq$ Infection $\sqcap \exists$hasCausalAgent.x_2	19,250 **3,073**	102.37 **2.69**	2,816
NAMEDLigament $\sqsubseteq$ NAMEDInternalBodyPart $\sqcap\ x_1$ $x_1 \sqsubseteq \exists$hasShapeAnalagousTo.$x_2 \sqcap \exists y$.linear	16,135 **197**	7.68 **1.12**	51
$x_1 \sqsubseteq$ NonNormalCondition $y_1 \sqsubseteq$ ModifierAttribute Bacterium $\sqsubseteq \exists y_1.x_2$ $y_2 \sqsubseteq$ StatusAttribute $x_2 \sqsubseteq$ AbstractStatus $x_1 \sqsubseteq \exists y_2$.Status	1,883 **1,883**	0.67 **0.80**	4,392

GALEN Queries: As expected, an increase in the number of variables within an axiom template leads to a significant increase in the query execution time because the number of mappings that have to be checked grows exponentially in the number of variables. This can, in particular, be observed from the difference in execution time between the first two queries, where it is also evident that the use of the hierarchy exploitation optimisation leads to a decrease in execution time of up to two orders of magnitude. In the last query, the hierarchy exploitation optimisation does not improve the execution time since the variables on which the hierarchy optimisation can be applied, are already bound when it comes to the evaluation of the complex templates. Hence, the running times with and without the hierarchy exploitation are similar. The number of consistency checks for this query is significantly lower than the number of answers because the overall results are computed by taking the cartesian products of the results for the two connected components of the query. Although our optimisations can significantly improve the query execution time, the required time can still be quite high. In practice, it is, therefore, advisable to add as many restrictive axiom templates (axiom templates which require only cache lookups) for query variables as possible. For example, the addition of $y \sqsubseteq$ Shape to the forth query reduces the runtime from 1.12 s to 0.65 s.

FBbt_XP Queries: We observe that in some queries the effect of the hierarchy exploitation is more profound than in others. More precisely, the smaller the ratio of the result size to the number of consistency checks without the hierarchy optimisation, the more pronounced is the effect when enabling this optimisation. In other words, when more tested mappings are indeed solutions, one can prune fewer parts of the hierarchy since pruning can only be performed when we find a non-solution. For the second query, we even observe a slight increase in running time when the hierarchy optimisation is used. This is because the optimisation can only prune few candidate mappings, which does not outweigh the overhead caused by maintaining information

Table 2. Queries (with $x_{(i)}$ a concept and y a role variable) and evaluation times in seconds for the FBbt_XP knowledge base with (bold) and without the hierarchy exploitation

Query	Consistency Checks	Time	Answers
$x_1 \sqsubseteq \forall$part_of.x_2	151,683	44.13	7,243
$x_1 \sqsubseteq$ FBbt_00005789	**11,262**	**5.64**	
$y \sqsubseteq$ part_of	14,446	37.38	7,224
$x \sqsubseteq \forall y$.FBbt_00001606	**12,637**	**39.20**	
$x \sqsubseteq \exists y$.FBbt_00025990	72,230	357.59	188
$y \sqsubseteq$ overlaps	**54,186**	**252.41**	
FBbt_00001606 $\sqsubseteq \exists y.x$	166,129	486.81	68
	1,335	**17.03**	
FBbt_00001606 $\sqsubseteq \exists y.x$	21,669	19.68	0
$y \sqsubseteq$ develops_from	**3**	**0.01**	
$x_1 \sqsubseteq$ FBbt_00001884	43,338	183.66	43,338
$y \sqsubseteq$ part_of	**32,490**	**152.38**	
$x_2 \sqsubseteq \exists y.x_1 \sqcap x_3$			

about which hierarchy parts have already been tested. For the last query, the number of consistency checks with the hierarchy optimisation is less than the result size ($32,490$ versus $43,338$) since only the third axiom template of the query requires consistency checks, whereas bindings for y and x_1 can be looked up from the cached concept and role hierarchy.

5 Conclusion

In the current paper we presented an approach for using the TBox of a knowledge base to optimise SPARQL queries evaluated using the OWL Direct Semantics entailment regime. For conjunctive instance 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. Conditions were defined for this purpose which identify the cases in which such a reduction is achieved. For queries that go beyond conjunctive instance queries we provide a highly effective and frequently applicable optimisation which prunes the number of candidate solutions that have to be checked by exploiting the concept and role hierarchy. One can, usually, assume that these hierarchies are computed before a system accepts queries. Our empirical evaluation shows that this optimisation can reduce the query evaluation times 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., Giacomo, G.D., 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)
3. Glimm, B., Horrocks, I., Motik, B., Shearer, R., Stoilos, G.: A novel approach to ontology classification. Journal of Web Semantics: Science, Services and Agents on the World Wide Web 14, 84–101 (2012), special Issue on Dealing with the Messiness of the Web of Data
4. Glimm, B., Ogbuji, C. (eds.): SPARQL 1.1 Entailment Regimes. W3C Recommendation (21 March 2013), available at http://www.w3.org/TR/sparql11-entailment/
5. Haarslev, V., Möller, R.: Racer system description. In: Gor, R., Leitsch, A., Nipkow, T. (eds.) Proceedings of the 1st International Joint Conference on Automated Reasoning (IJCAR'01). LNCS, vol. 2083, pp. 701–705. Springer (2001)
6. Haarslev, V., Möller, R., Wessel, M.: Querying the semantic web with Racer + nRQL. In: Proceedings of the KI-2004 International Workshop on Applications of Description Logics (2004)
7. Harris, S., Seaborne, A. (eds.): SPARQL 1.1 Query Language. W3C Recommendation (21 March 2013), available at http://www.w3.org/TR/sparql11-query/
8. Horridge, M., Bechhofer, S.: The OWL API: A Java API for working with OWL 2 ontologies. In: Patel-Schneider, P.F., Hoekstra, R. (eds.) Proceedings of the OWLED 2009 Workshop on OWL: Experiences and Directions. CEUR Workshop Proceedings, vol. 529. CEUR-WS.org (2009)
9. Hustadt, U., Motik, B., Sattler, U.: Reducing $\mathcal{SHIQ}^-$ description logic to disjunctive datalog programs. In: Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR'04). pp. 152–162. AAAI Press (2004)
10. Johnson, D.S., Klug, A.C.: Testing containment of conjunctive queries under functional and inclusion dependencies. J. Comput. Syst. Sci. 28(1), 167–189 (1984)
11. Kollia, I., Glimm, B.: Optimizing SPARQL Query Answering over OWL Ontologies. Accepted at Journal of Artificial Intelligence Research (2013)
12. Kollia, I., Glimm, B.: Cost based query ordering over OWL ontologies. In: Proceedings of the 11th International Semantic Web Conference (ISWC 2012). Lecture Notes in Computer Science, vol. 7649, pp. 231–246. Springer-Verlag (2012)
13. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The combined approach to query answering in DL-Lite. In: Lin, F., Sattler, U. (eds.) Proceedings of the 12th International Conference on Principles of Knowledge Representation and Reasoning (KR'10). AAAI Press (2010)
14. Motik, B., Patel-Schneider, P.F., Cuenca Grau, B. (eds.): OWL 2 Web Ontology Language: Direct Semantics. W3C Recommendation (27 October 2009), available at http://www.w3.org/TR/owl2-direct-semantics/
15. Pan, J.Z., Thomas, E., Zhao, Y.: Completeness guaranteed approximation for owl dl query answering. In: Proceedings of the 22nd International Workshop on Description Logics (DL2009) (2009)
16. Pérez-Urbina, H., Motik, B., Horrocks, I.: Tractable query answering and rewriting under description logic constraints. Journal of Applied Logic 8(2), 186–209 (2010)
17. Ren, Y., Pan, J.Z., Zhao, Y.: Soundness preserving approximation for tbox reasoning. In: Proceedings of the 25th AAAI Conference (AAAI2010) (2010)

18. Sirin, E., Cuenca Grau, B., Parsia, B.: From wine to water: Optimizing description logic reasoning for nominals. In: Doherty, P., Mylopoulos, J., Welty, C.A. (eds.) Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR'06). pp. 90–99. AAAI Press (2006)
19. Sirin, E., Parsia, B.: SPARQL-DL: SPARQL query for OWL-DL. In: Golbreich, C., Kalyanpur, A., Parsia, B. (eds.) Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions. CEUR Workshop Proceedings, vol. 258. CEUR-WS.org (2007)
20. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Journal of Web Semantics 5(2), 51–53 (2007)
21. Thomas, E., Pan, J.Z., Ren, Y.: TrOWL: Tractable OWL 2 reasoning infrastructure. In: Proceedings of the Extended Semantic Web Conference (ESWC'10) (2010)
22. Tsarkov, D., Horrocks, I., Patel-Schneider, P.F.: Optimizing terminological reasoning for expressive description logics. Journal of Automated Reasoning 39(3), 277–316 (2007)

A Proofs

Lemma (1). *Algorithm* intersecQans *is an approximate query answering algorithm.*

Proof (Proof of Lemma 1). We prove the lemma by induction on the length of the query q given as input to the algorithm $\mathsf{intersecQans}(\mathcal{K}, q)$. For $q = \{\mathsf{at}\}$ the two conditions of the definition of approximate query answering algorithms are satisfied. Indeed,

1. if $\mu \in K[q]$, i.e., $\mu \in K[\mathsf{at}]$ then $\mathcal{K} \models \mu(at)$ (since $\mathsf{inst}(\mathcal{K}, \mathsf{at})$ is an approximate instance retrieval algorithm), which means that $\mathcal{K} \models \mu(q)$.
2. for each total function μ from $Var(q)$ to individual names from $\mathcal{K}$ such that $\mu \in \mathsf{ans}(\mathcal{K}, q)$, i.e., $\mu \in \mathsf{ans}(\mathcal{K}, \mathsf{at})$, i.e., $\mathcal{K} \models \mu(\mathsf{at})$, it holds $\mu \in K[\mathsf{at}] \cup P[\mathsf{at}]$ (since $\mathsf{inst}(\mathcal{K}, \mathsf{at})$ is an approximate instance retrieval algorithm), i.e., $\mu \in K[q] \cup P[q]$.

Let $q' = q \cup \{\mathsf{at}\}$ and let us assume that $\mathsf{intersecQans}(\mathcal{K}, q)$ is an approximate query answering algorithm. We prove that $\mathsf{inst}(\mathcal{K}, q')$ is also an approximate query answering algorithm by showing that the two conditions of Definition 3 hold for q'.

1. If $\mu \in K[q']$, this means that $\mu_{|Var(q)} \in K[q]$ and $\mu_{|Var(\mathsf{at})} \in K[\mathsf{at}]$ (it can easily be seen from the construction $K[q']$ in $\mathsf{intersecQans}(\mathcal{K}, q')$). By induction hypothesis $\mu_{|Var(q)} \in \mathsf{ans}(\mathcal{K}, q)$ and, since $\mathsf{inst}(\mathcal{K}, \mathsf{at})$ is an approximate instance retrieval algorithm, $\mathcal{K} \models \mu_{|Var(\mathsf{at})}(\mathsf{at})$. Since $\mu = \mu_{|Var(q)} \bowtie \mu_{|Var(\mathsf{at})}$, it holds $\mu \in \mathsf{ans}(\mathcal{K}, q')$.
2. For each total function μ from $Var(q)$ to individual names from $\mathcal{K}$ such that $\mu \in \mathsf{ans}(\mathcal{K}, q')$, it holds $\mu_{|Var(q)} \in \mathsf{ans}(\mathcal{K}, q)$ and $\mathcal{K} \models \mu_{|Var(\mathsf{at})}(\mathsf{at})$. By the induction hypothesis, $\mu_{|Var(q)} \in K[q] \cup P[q]$ and $\mu_{|Var(\mathsf{at})} \in K[\mathsf{at}] \cup P[\mathsf{at}]$. It holds $\mu = \mu_{|Var(q)} \bowtie \mu_{|Var(\mathsf{at})}$. We distinguish between the following cases:
   - if $\mu_{|Var(q)} \in K[q]$ and $\mu_{|Var(\mathsf{at})} \in K[\mathsf{at}]$, then $\mu \in K[q']$
   - if $\mu_{|Var(q)} \in K[q]$ and $\mu_{|Var(\mathsf{at})} \in P[\mathsf{at}]$, then $\mu \in P[q']$
   - if $\mu_{|Var(q)} \in P[q]$ and $\mu_{|Var(\mathsf{at})} \in K[\mathsf{at}]$, then $\mu \in P[q']$
   - if $\mu_{|Var(q)} \in P[q]$ and $\mu_{|Var(\mathsf{at})} \in P[\mathsf{at}]$, then $\mu \in P[q']$

   We see from the above that in any case $\mu \in K[q'] \cup P[q']$.

□

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

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

Proof (Proof of Lemma 2). For the first claim, according to Lemma 1, $\mathsf{intersecQans}(\mathcal{K}, q)$ is an approximate query answering algorithm. Hence, the first condition on approximate query answering algorithms is satisfied. For the second condition and in contrary to what is to be shown, assume there is some μ such that $\mu \in \mathsf{ans}(\mathcal{K}, q)$, but $\mu \notin K[q] \cup \{\mu \in P[q] \mid \mu_{|Var(\mathsf{at})} \in (K[\mathsf{at}] \cup P[\mathsf{at}])\}$, i.e., $\mu_{|Var(\mathsf{at})} \notin K[\mathsf{at}] \cup P[\mathsf{at}]$. By Definition 4 and since at restricts $P[q]$, we have $Var(\mathsf{at}) \subseteq Var(q)$ and all variables from at are in the domain of μ. Since $\mathsf{inst}(\mathcal{K}, \mathsf{at})$ is an approximate instance retrieval algorithm,

this means $\mathcal{K} \not\models \mu_{|Var(\mathsf{at})}(\mathsf{at})$, but this means that $\mu \notin \mathsf{ans}(\mathcal{K}, q')$, which contradicts $\mathsf{ans}(\mathcal{K}, q) = \mathsf{ans}(\mathcal{K}, q')$.

The second claim follows 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 4 and since at restricts q. □

Lemma (3). *The extended query $\hat{q}$ created as in Definition 5 is unique.*

Proof (Proof of Lemma 3). It is obvious that the extended query $\hat{q}$ does not depend on the function f that is used for its construction since the canonical ABoxes and consequently the extended canonical ABoxes produced w.r.t. different functions f are isomorphic to each other, i.e., identical modulo renaming of individuals. □

Theorem (1). *Let $\mathcal{T}$ be a TBox, q a query, and $\hat{q}$ the extended query as in Definition 5. 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 (Proof of Theorem 1).

We want to prove that for any ABox $\mathcal{A}$, $\mathsf{ans}(\langle \mathcal{T}, \mathcal{A} \rangle, q) = \mathsf{ans}(\langle \mathcal{T}, \mathcal{A} \rangle, \hat{q})$, i.e., $\mathsf{ans}(\langle \mathcal{T}, \mathcal{A} \rangle, q) \subseteq \mathsf{ans}(\langle \mathcal{T}, \mathcal{A} \rangle, \hat{q})$ and $\mathsf{ans}(\langle \mathcal{T}, \mathcal{A} \rangle, \hat{q}) \subseteq \mathsf{ans}(\langle \mathcal{T}, \mathcal{A} \rangle, q)$. From Definition 5 it is easily seen that $Var(q) = Var(\hat{q})$ and since q has more atoms (is more restricting) than q, it holds $\mathsf{ans}(\langle \mathcal{T}, \mathcal{A} \rangle, \hat{q}) \subseteq \mathsf{ans}(\langle \mathcal{T}, \mathcal{A} \rangle, q)$. We will now show that $\mathsf{ans}(\langle \mathcal{T}, \mathcal{A} \rangle, q) \subseteq \mathsf{ans}(\langle \mathcal{T}, \mathcal{A} \rangle, \hat{q})$. Let μ be a mapping such that $\mu \in \mathsf{ans}(\langle \mathcal{T}, \mathcal{A} \rangle, q)$, i.e., $\langle \mathcal{T}, \mathcal{A} \rangle \models \mu(q)$. From Definition 5 it is obvious that for any total function f from $Var(q)$ to a set of individual names from N_I, $\langle \mathcal{T}, \mathcal{A}^f_q \rangle \models \hat{\mathcal{A}}^f_q$. Hence, $\langle \mathcal{T}, \mathcal{A}^\mu_q \rangle \models \hat{\mathcal{A}}^\mu_q$ or $\langle \mathcal{T}, \mu(q) \rangle \models \mu(\hat{q})$ which means $\langle \mathcal{T}, \mathcal{A} \cup \mu(q) \rangle \models \mu(\hat{q})$ due to the monotonicity property of description logics, which means that $\langle \mathcal{T}, \mathcal{A} \rangle \models \mu(\hat{q})$ (since $\langle \mathcal{T}, \mathcal{A} \rangle \models \mu(q)$), i.e., $\mu \in \mathsf{ans}(\langle \mathcal{T}, \mathcal{A} \rangle, \hat{q})$.

□

Theorem (2). *Let $\mathcal{K}$ be a knowledge base, A, B concept names such that $\mathcal{K} \models A \sqsubseteq B$, C_1, C_2 concept templates, $C = \neg C_1 \sqcup C_2$, $x \in V_C$ a concept variable occurring in C, and μ a mapping that covers all variables of C apart from x.*

1. *If $\mathsf{pol}_c(x, C) = \mathsf{pos}$ and $\mathcal{K} \not\models (C_1 \sqsubseteq C_2)_{\mu(x)=B}$, then $\mathcal{K} \not\models (C_1 \sqsubseteq C_2)_{\mu(x)=A}$.*
2. *If $\mathsf{pol}_c(x, C) = \mathsf{neg}$ and $\mathcal{K} \not\models (C_1 \sqsubseteq C_2)_{\mu(x)=A}$, then $\mathcal{K} \not\models (C_1 \sqsubseteq C_2)_{\mu(x)=B}$.*

Proof (Proof Sketch of Theorem 2). The claim can be shown by induction on the structure of the concept template C. We only consider the case of $\mathsf{pol}_c(x, C) = \mathsf{pos}$; the case of $\mathsf{pol}_c(x, C) = \mathsf{neg}$ is analogous. It suffices to show (in contrapositive form) for some model $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ of $\mathcal{K}$ and some element $\delta \in \Delta^\mathcal{I}$, if $\delta \in (C_{\mu(x)=A})^\mathcal{I}$, then $\delta \in (C_{\mu(x)=B})^\mathcal{I}$.

For the base case, $C = x$, x occurs positively in C. Now, if $\delta \in (x_{\mu(x)=A})^\mathcal{I}$, then $\delta \in A^\mathcal{I}$ and, hence, $\delta \in B^\mathcal{I}$ since $\mathcal{K} \models A \sqsubseteq B$ by assumption. Hence, $\delta \in (x_{\mu(x)=B})^\mathcal{I}$.

We show the induction step for $C = \exists r.D$. The other cases are analogous. We assume $\delta \in ((\exists r.D)_{\mu(x)=A})^\mathcal{I}$, then δ has at least one r-successor, say δ', that is an instance of $D_{\mu(x)=A}$. Since $\mathcal{K} \models A \sqsubseteq B$ and by induction hypothesis, $\delta' \in D_{\mu(x)=B}$. Hence, $\delta \in (\exists r.(D_{\mu(x)=B}))^\mathcal{I} = ((\exists r.D)_{\mu(x)=B})^\mathcal{I}$. Note that the case $C = (= n\ r.D)$ cannot occur since the polarity of x in C is always positive and negative and $\mathsf{pol}_c(x, C)$ is undefined. □

Theorem (3). *Let $\mathcal{K}$ be a knowledge base, r, s role names such that $\mathcal{K} \models r \sqsubseteq s$, C_1, C_2 concept templates, $C = \neg C_1 \sqcup C_2$, $x \in V_R$ a role variable occurring in C, and μ a mapping that covers all variables of C apart from x.*

1. *If $\mathsf{pol}_r(x, C) = \mathsf{pos}$ and $\mathcal{K} \not\models (C_1 \sqsubseteq C_2)_{\mu(x)=s}$, then $\mathcal{K} \not\models (C_1 \sqsubseteq C_2)_{\mu(x)=r}$.*
2. *If $\mathsf{pol}_r(x, C) = \mathsf{neg}$ and $\mathcal{K} \not\models (C_1 \sqsubseteq C_2)_{\mu(x)=r}$, then $\mathcal{K} \not\models (C_1 \sqsubseteq C_2)_{\mu(x)=s}$.*

Proof (Proof Sketch of Theorem 3). The claim can be shown by induction on the structure of the concept template C. We only consider the case of $\mathsf{pol}_r(x, C) = \mathsf{pos}$; the case of $\mathsf{pol}_r(x, C) = \mathsf{neg}$ is analogous. It suffices to show (in contrapositive form) for some model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\mathcal{K}$ and some element $\delta \in \Delta^{\mathcal{I}}$, if $\delta \in (C_{\mu(x)=r})^{\mathcal{I}}$, then $\delta \in (C_{\mu(x)=s})^{\mathcal{I}}$.

We have several base cases here, but only present the case for concept templates of the form $C = \exists x.D$ with x not occurring in D. The cases for $C = \forall x.D$, $C = \geqslant n\ x.D$, $C = \leqslant n\ x.D$ with x not occurring in D are similar. Assume, $\delta \in ((\exists x.D)_{\mu(x)=r})^{\mathcal{I}}$, that is, $\delta \in (\exists r.\mu(D))^{\mathcal{I}}$. Then there is some $\delta' \in \Delta^{\mathcal{I}}$ such that $\langle \delta, \delta' \rangle \in r^{\mathcal{I}}$ and $\delta' \in \mu(D)^{\mathcal{I}}$. Since $\mathcal{K} \models r \sqsubseteq s$, we also have $\langle \delta, \delta' \rangle \in s^{\mathcal{I}}$ and, therefore, $\delta \in (\exists s.\mu(D))^{\mathcal{I}} = ((\exists x.D)_{\mu(x)=s})^{\mathcal{I}}$.

For the induction step, let $C = \exists x.D$ with x occurring in D, we also have $\mathsf{pol}_r(x, D) = \mathsf{pos}$. Now, if $\delta \in ((\exists x.D)_{\mu(x)=r})^{\mathcal{I}}$, then δ has at least one r-successor which is an instance of $D_{\mu(x)=r}$. Since $\mathcal{K} \models r \sqsubseteq s$ and by induction hypothesis, δ has at least one s-successor that is an instance of $D_{\mu(x)=s}$. Hence, $\delta \in ((\exists x.D)_{\mu(x)=s})^{\mathcal{I}}$. The case of $C = \exists p.D$ with p a role and x occurring in D is analogous and so are the further induction steps. □