Improving semantic query answering

Norbert Kottmann and Thomas Studer

Institut für Informatik und angewandte Mathematik,
Universität Bern, Neubrückstrasse 10, CH-3012 Bern, Switzerland,
{kottmann,tstuder}@iam.unibe.ch

Abstract. The retrieval problem is one of the main reasoning tasks for knowledge base systems. Given a knowledge base K and a concept C, the retrieval problem consists of finding all individuals a for which K logically entails $C(a)$. We present an approach to answer retrieval queries over (a restriction of) OWL ontologies. Our solution is based on reducing the retrieval problem to a problem of evaluating an SQL query over a database constructed from the original knowledge base. We provide complete answers to retrieval problems. Still, our system performs very well as is shown by a standard benchmark.

1 Introduction

Over the last decade, ontologies left the realm of academia and became an important technology in many domains. However, in order to be of practical use for full-fledged applications, tools and techniques that can deal with huge amounts of (ontological) information are needed.

Relational databases are one of the well-established cornerstones for systems managing large data loads. In this paper, we present a method to solve the ontological retrieval problem based on a relational database system. Our implementation shows that this provides an efficient and scalable solution for the retrieval problem.

An ontology defines the terms used to describe and represent an area of knowledge [7]. It consists of the definitions for the basic concepts of a domain and their relationships. These definitions and relations are formulated in a so-called ontology language which should be not only understandable for humans but also machine readable, hence supporting automatic knowledge processing. The W3C defined the ontology language OWL for applications in the semantic web. However, OWL also became the language of choice for many other applications in the area of knowledge representation and reasoning.

One of the main reasoning tasks for knowledge base systems is the so-called *retrieval problem* [2]. Let us illustrate this problem by an example.

Assume a zoological ontology defines the following:

(1) A carnivore is an animal that eats only animals.
(2) A lion is a carnivore.
(3) A lion eats gnus.

Further, assume that this ontology has been loaded into a knowledge base system. This system will answer the query *show me all animals* as follows:

1. A lion is an animal since a lion is a carnivore (2) and every carnivore is an animal (1).
2. A gnu is an animal since a lion is a carnivore (2), everything that is eaten by a carnivore is an animal (1), and gnus are eaten by lions (3).

The abstract definition of the retrieval problem reads as follows: given a knowledge base K and a concept description C, find all individuals a such that K logically entails $C(a)$. That is given K and C, look for all individuals a such $C(a)$ is a logical consequence of K. There is a trivial algorithm for this problem, namely to test for each individual occurring in K whether it satisfies the concept C or not. This approach has the advantage that it provides (almost) complete reasoning for quite expressive knowledge representation languages. However, if large data sets have to be treated, then for efficiency reasons, one may need to turn to another approach.

It is possible to extend relational database systems to support storing and querying OWL [4] data as follows: when data is loaded into the database, the system precomputes the subsumption hierarchy and stores also the statements inferred from this hierarchy. Prominent projects following this approach are DLDB [10] and its successor HAWK[1]. Queries to the knowledge base can then be translated to standard SQL queries that are evaluated over the relational representation of the knowledge base. This has the advantage that all the query optimization techniques provided by relational database systems can be used and it becomes possible to work with huge datasets.

However, DLDB and HAWK often do not give complete answers to queries. We overcome this problem by identifying the description logic pos-$\mathcal{ALE}$ which is the positive fragment of $\mathcal{ALE}$ with transitive and inverse roles. Based on pos-$\mathcal{ALE}$, we present an extension of relational database systems to support OWL retrieval queries which is sound and complete with respect to pos-$\mathcal{ALE}$.

[1] `http://swat.cse.lehigh.edu/downloads/`

The language of pos-$\mathcal{ALE}$ provides enough expressive power for many applications. It also suffices for the LUBM benchmark for OWL knowledge base systems [5]. We evaluate our system with this standard benchmark and compare our results with HAWK. The main observation is that our systems performs very well although we provide complete pos-$\mathcal{ALE}$ reasoning. For many queries we are even faster that HAWK (which is often not complete).

Another approach for querying ontologies with the use of an SQL engine has been presented in [1, 3]. There, the DL-Lite family of description logics is introduced. These languages are also well suited for the translation of description logic queries into SQL queries. However, DL-Lite languages are quite different from pos-$\mathcal{ALE}$. On one hand, pos-$\mathcal{ALE}$ features value restrictions and transitive roles which are both not included in DL-Lite. DL-Lite, on the other hand, supports functional restrictions and conjunctive queries which cannot be treated in pos-$\mathcal{ALE}$. Because of all these differences, we could not compare our approach with a DL-Lite system.

2 DL to DB Mapping

The concepts of pos-$\mathcal{ALE}$ are given as follows, where A is used for an *atomic concept*, S is an *atomic role*, R, T denote *roles*, and C, D stand for *concept descriptions*:

$$
\begin{array}{rll}
C, D \rightarrow & A \mid & \text{(atomic concept)} \\
& \top \mid & \text{(top)} \\
& C \sqcap D \mid & \text{(conjunction)} \\
& \forall R.C \mid & \text{(value restriction)} \\
& \exists R.C & \text{(full existential quantification)} \\
R \rightarrow & S \mid & \text{(atomic role)} \\
& S^{-} & \text{(inverse role).}
\end{array}
$$

Additionally, we consider a set R^{+} of transitive roles. A TBox T contains concept inclusions $C \sqsubseteq D$ as well as role inclusions $R \sqsubseteq T$. An ABox A contains concept assertions $C(a)$ and role assertions $R(a, b)$. A knowledge base K is the union of a TBox and an ABox.

We make use of the standard semantics for description logics [2]. Accordingly, we write $\mathsf{K} \models C(a)$ if every model of K is a model of $C(a)$.

Our aim is to build a *completion* A^{*} of the ABox A such that is possible to answer arbitrary pos-$\mathcal{ALE}$ retrieval queries by only querying *atomic* concept and roles in A^{*}. Assume DB_{K} is the subset of all atomic

concept and role assertion of such a completed ABox (stemming from an initial knowledge base K). Then we write $\mathsf{DB}_\mathsf{K} \models_{\mathsf{DB}} C(a)$ if a is in the answer to the retrieval query C when it is evaluated over DB_K. This evaluation is inductively defined as follows.

1. $\mathsf{DB}_\mathsf{K} \models_{\mathsf{DB}} A(a)$ if $A(a) \in \mathsf{DB}_\mathsf{K}$
2. $\mathsf{DB}_\mathsf{K} \models_{\mathsf{DB}} R(a,b)$ if $R(a,b) \in \mathsf{DB}_\mathsf{K}$
3. $\mathsf{DB}_\mathsf{K} \models_{\mathsf{DB}} C \sqcap D(a)$ if $\mathsf{DB}_\mathsf{K} \models_{\mathsf{DB}} C(a)$ and $\mathsf{DB}_\mathsf{K} \models_{\mathsf{DB}} D(a)$
4. $\mathsf{DB}_\mathsf{K} \models_{\mathsf{DB}} \forall R.C(a)$ if $\mathsf{DB}_\mathsf{K} \models_{\mathsf{DB}} C(\forall_{R,a})$
5. $\mathsf{DB}_\mathsf{K} \models_{\mathsf{DB}} \exists R.C(a)$ if there exists a constant b with $\mathsf{DB}_\mathsf{K} \models_{\mathsf{DB}} C(b)$ and $\mathsf{DB}_\mathsf{K} \models_{\mathsf{DB}} R(a,b)$

The constants $\forall_{R,a}$ are special individual terms introduced in the completion process in order to correctly answer queries which involve value restrictions.

Remark 1. The above definition makes it possible to formulate retrieval queries over DB_K using standard SQL.

Before we can perform the completion algorithm which computes the relational representation of a knowledge base K, we have to normalize K, that is replace every occurrence of $\forall R.(C \sqcap D)$ with $\forall R.C \sqcap \forall R.D$.

Then, the *precompletion* A' of A is built by applying the following rules to an ABox A until a fixed point is reached. Of course, these rule are reminiscent of the tableau construction for description logics with transitive and inverse roles, see for instance [8].

1. $\top(a) \in \mathsf{X}$ if a occurs in X
2. $C(a) \in \mathsf{X}$ and $D(a) \in \mathsf{X}$ if $C \sqcap D(a) \in \mathsf{X}$
3. $C(x) \in \mathsf{X}$ and $R(a,x) \in \mathsf{X}$ for a new x if $\exists R.C(a) \in \mathsf{X}$ and no such x exists yet
4. $C(a) \in \mathsf{X}$ if $\forall R.C(b) \in \mathsf{X}$ and $R(b,a) \in \mathsf{X}$ for some b
5. $C(\forall_{R,a}) \in \mathsf{X}$ if $\forall R.C(a) \in \mathsf{X}$
6. $C(\forall_{R,a}) \in \mathsf{X}$ if $\forall R.C(b) \in \mathsf{X}$, $R(b,a) \in \mathsf{X}$, and $R \in \mathsf{R}^+$
7. $R(a,b) \in \mathsf{X}$ if $T(a,b) \in \mathsf{X}$ and $T \sqsubseteq R \in \mathsf{T}$
8. $R^-(a,b) \in \mathsf{X}$ if $R(b,a) \in \mathsf{X}$ where we set $(R^-)^- := R$
9. $R(a,c) \in \mathsf{X}$ if $R \in \mathsf{R}^+$, $R(a,b) \in \mathsf{X}$, and $R(b,c) \in \mathsf{X}$ for some b

The only part of K that is not taken into account in the build up of the precompletion are the concept inclusions present in the TBox. In order to treat them properly, we have to apply the following algorithm.

Algorithm 1 Procedure for building the completion of an ABox

Input: ABox X and TBox T
Output: Completion of the ABox

$\mathsf{Y} \longleftarrow \emptyset$
repeat
  $\mathsf{X} \longleftarrow \mathsf{X} \cup \mathsf{Y}$
  $\mathsf{X} \longleftarrow$ precompletion of X
  $\mathsf{Y} \longleftarrow \{C(a) : \text{there exists a concept } D \text{ such that } \mathsf{DB}_\mathsf{X} \models D(a) \text{ and } D \sqsubseteq C \in \mathsf{T}\}$
until $\mathsf{Y} = \emptyset$
return X

That is starting from an initial ABox A_1 we build the precompletion $\mathsf{A'}_1$. Then we add all assertions implied by inclusion axioms yielding an ABox A_2. We have to precomplete this ABox resulting in $\mathsf{A'}_2$. Again, the inclusion axioms may imply new assertions which gives us an ABox A_3. This process eventually stops which provides the completion A^* of A_1

Example 1. Consider the zoological ontology given in the introduction. We have

$$\mathsf{T} := \{\mathsf{Carnivore} \sqsubseteq \mathsf{Animal} \sqcap \forall \mathsf{Eats.Animal}\}$$

and

$$\mathsf{A} := \{\mathsf{Carnivore}(\mathsf{lion}), \mathsf{Eats}(\mathsf{lion}, \mathsf{gnu})\}.$$

Let $\mathsf{A}_1 := \mathsf{A}$. Building the precompletion of A_1 does not give any new assertions. The second step deals with the concept hierarchy. That yields

$$\mathsf{A}_2 = \mathsf{A}_1 \cup \{\mathsf{Animal} \sqcap \forall \mathsf{Eats.Animal}(\mathsf{lion})\}.$$

Building the precompletion of A_2 gives us

$$\mathsf{Animal}(\mathsf{lion}), \forall \mathsf{Eats.Animal}(\mathsf{lion}), \mathsf{Animal}(\mathsf{gnu}) \in \mathsf{A'}_2.$$

Since no new individuals have been added to $\mathsf{Carnivore}$, the second step dealing with the concept hierarchy does not result in additional assertions. Thus, we reached a fixed point and we have $\mathsf{A'}_2 = \mathsf{A}^*$.

If we start from a finite knowledge base K, then by a cardinality argument we easily can see that a fixed point is reached after finitely many steps.

Theorem 1. *If* K *is a finite knowledge base, then also its completion* K^* *is finite.*

Proof. First observe, that only finitely many new individual constants are needed in the course of the inductive built up of K^*. Consider the case for value restrictions. There, it may be that a constant of the form $\forall_{R,\forall_{\ldots},\forall_{T,a}}$ is introduced. However the role depth of such a new constant can at most be the role depth of the original knowledge base K. The same goes for the case of existential quantification. Therefore, only finitely many new constants have to be introduced. Moreover, individual constants are only added to subconcepts of concepts occurring in K and there are only finitely many such subconcepts. Hence, the fixed point will be reached after finitely many stages. □

The database instance DB_K of a knowledge base K consists of all atomic concept assertions $A(a)$ and role assertions $R(a, b)$ of the completion K^*.

Due to the interplay of $\forall$ and $\exists$, the size of DB_K can be exponential in the size of K [2]. However, in many practical applications this blow-up does not happen. For instance, our evaluation shows that in the LUBM benchmark the database size grows only linearly in the size of the original knowledge base.

In the sequel we show that query evaluation over DB_K is sound and complete.

Theorem 2 (Completeness). *Let* K *be a knowledge base,* C *be a concept description, and* a *be an individual constant. We have that*

$$\mathsf{K} \models C(a) \Longrightarrow \mathsf{DB}_\mathsf{K} \models_{\mathsf{DB}} C(a).$$

Proof. Assume $\mathsf{DB}_\mathsf{K} \not\models_{\mathsf{DB}} C(a)$. Completeness is easily established by constructing a canonical counter model $\mathcal{M}$ with $\mathcal{M} \models \mathsf{K}$ and $\mathcal{M} \not\models C(a)$. The only point that needs a bit of attention is the case when C is a value restriction $\forall R.D$. In this case we have to observe that it is always possible in pos-$\mathcal{ALE}$ to extend a given interpretation of R to falsify $\forall R.D(a)$. □

In order to prove soundness we need some auxiliary definitions.

Definition 1. *The* $*$ *unfolding of a concept assertion is given by:*

1. $C(a)^* := C(a)$ *if* a *is not of the form* $\forall_{R,b}$,
2. $C(a)^* := (\forall R.C(b))^*$ *if* $a = \forall_{R,b}$.

Definition 2. *The basis of an individual constant is given by:*

1. $\mathsf{basis}(a) := \mathsf{basis}(b)$ *if* $a = \forall_{R,b}$,
2. $\mathsf{basis}(a) := a$ *otherwise.*

Theorem 3. *Let* K *be a knowledge base,* C *be a concept description, and* a *be an individual constant such that* $\mathsf{basis}(a)$ *occurs in* K*. We have that*

$$\mathsf{DB}_{\mathsf{K}} \models_{\mathsf{DB}} C(a) \Longrightarrow \mathsf{K} \models C(a)^*.$$

Proof. By induction on the structure of C. We show only the case when C is $\forall R.D$. Then $\mathsf{DB}_{\mathsf{K}} \models_{\mathsf{DB}} D(\forall_{R,a})$. By the induction hypothesis we get $\mathsf{K} \models D(\forall_{R,a})^*$. That is $\mathsf{K} \models \forall R.D(a)^*$ by the previous definition. □

As a corollary we obtain:

Corollary 1 (Soundness). *Let* K *be a knowledge base,* C *be a concept description, and* a *be an individual constant. We have that*

$$\mathsf{DB}_{\mathsf{K}} \models_{\mathsf{DB}} C(a) \Longrightarrow \mathsf{K} \models C(a).$$

3 Evaluation

To show the applicability of our approach, we developed a system called REAL based on pos-$\mathcal{ALE}$ and the completion procedure presented above, see [9]. We evaluated our implementation with the Lehigh University Benchmark [5]. This is a standard benchmark for expressive semantic web knowledge base systems. We compared the performance of our system with that of HAWK which also follows a completion approach. We used a 3 GHz Pentium 4, 2 GB RAM, and a PostgreSQL DB to run the tests.

The overall setup of the benchmark is shown in Figure 3. The benchmark system contains

1. an OWL ontology modeling a university domain,
2. a data generator (UBA) creating datasets (ABoxes) of different size,
3. a performance test application (UBT) which runs the benchmark, and
4. a set of test queries.

The system under consideration have to provide a benchmark interface (BM) which is called by the test application. Note that HAWK uses an external reasoner (RacerPro, see [6]) for some initial computations in the loading process of an ontology. The detailed settings and all results of the evaluation can be found in [9].

We tested four different datasets called 01, 05, 10, and 20. The smallest dataset (01) contains about 100'000 triples which equals a file size of 8 MB whereas the largest dataset (20) counts more than 2'700'000 triples with a total size of 234 MB.

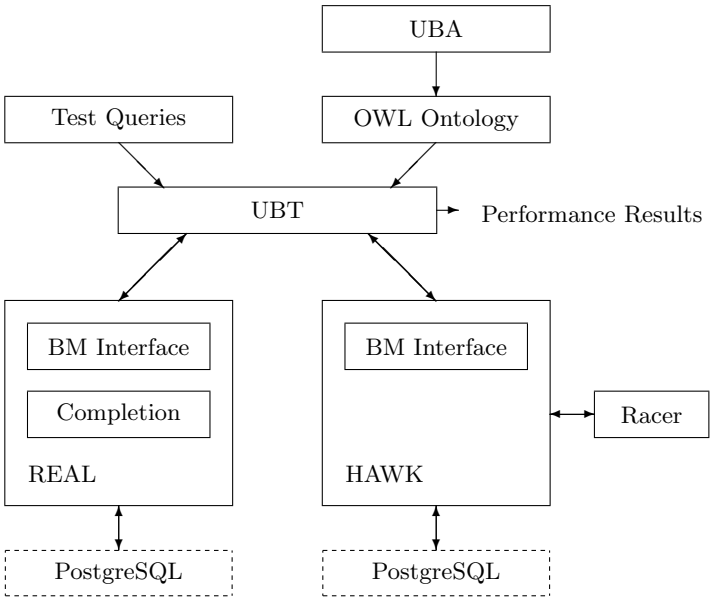

Fig. 1. Benchmark setup.

The benchmark consists of 14 queries which we issued against the four datasets in both systems. First note that our system provides complete answers to all queries whereas HAWK only provides complete answers to the queries 1,2,3,4, and 14. For all datasets, the answer times to the queries are shown in Figure 2. We find that although we provide complete answers to all queries, our system often perform even better than HAWK.

Our system also scales up very well. For many queries we have linear (sometimes almost constant) behavior of the answering time with respect to the number of triples. See Figure 3 for two typical examples.

For the queries 2, 9, and 12 we need much more time than HAWK and our system does not show a good scaling behavior. These queries need a lot of joins and we have not yet found an optimal database configuration to better support such queries in our approach. However note that HAWK does not provide complete answers to 9 and 12. Still query 2 shows that there is room for improvement.

4 Concluding Remarks

We identified pos-$\mathcal{ALE}$, a description logic which can easily be represented in a relational setting. This leads to an extension of relational databases

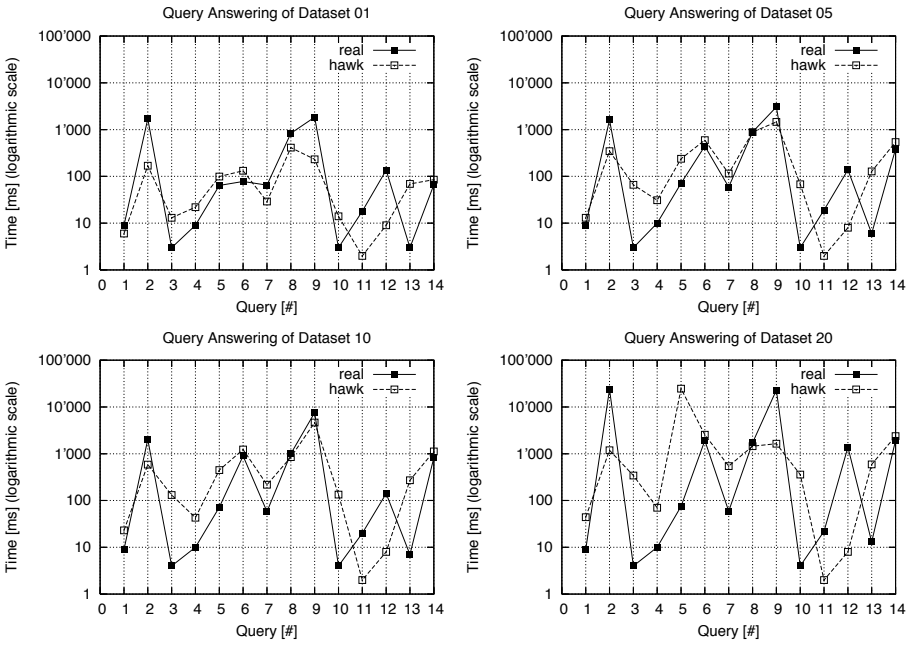

Fig. 2. All datasets with the response time of each query.

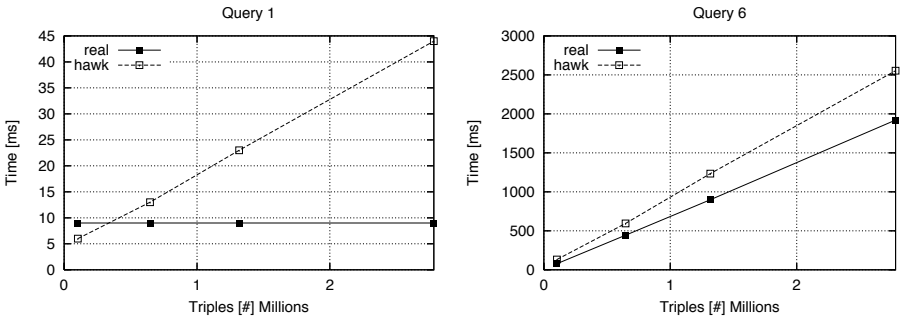

Fig. 3. Response time of queries 1 and 6 in relation to the number of triples in the database.

which supports semantic web queries. So far, such extensions often did not provide complete answers to retrieval problems. Our relational representation provides sound and complete query answering with respect to pos-$\mathcal{ALE}$. The evaluation of our implementation with the LUBM benchmark showed that our approach is suitable for practical applications. In particular, it exhibits good scaling properties. However, the tests also showed that we still need better support for queries that involve many join operations.

Scalability is not the only feature which makes our approach valuable for practical applications. The use of a classical database system has the additional advantage that all the features provided by the database may be employed. For example, access control for the ontological system can be implemented based on the privileges and rights system the database provides.

Acknowledgments

We would like to thank Yuanbo Guo for his support on LUBM.

References

1. A. Acciarri, D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, and R. Rosati. Quonto: Querying ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 1670–1671, 2005.
2. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook*. Cambridge, 2003.
3. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. DL-Lite: Tractable description logics for ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 602–607, 2005.
4. M. Dean and G. Schreiber. OWL web ontology language reference, 2004.
5. Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, 3:158–182, 2005.
6. V. Haarslev and R. Möller. Racer system description. In R. Goré, A. Leitsch, and T. Nipkow, editors, *International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, Siena, Italy*, pages 701–705. Springer-Verlag, 2001.
7. J. Heflin. OWL web ontology language use cases and requirements, 2004. Available at `http://www.w3c.org/TR/webont-req/`.
8. I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. *Journal of Logic and Computation*, 9(3):385–410, 1999.
9. N. Kottmann. Description logic query answering with relational databases. Master's thesis, University of Bern, 2006.
10. Z. Pan and J. Heflin. DLDB: Extending relational databases to support semantic web queries. In *Workshop on Practical and Scaleable Semantic Web Systems, ISWC 2003*, pages 109–113, 2003.