Semantic Index: Scalable Query Answering without Forward Chaining or Exponential Rewritings

Mariano Rodríguez-Muro and Diego Calvanese

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

Entailment regimes add support for rich inferences in SPARQL 1.1. This greatly facilitates the use of reasoning in applications. In this context, one special interest of the Semantic Web (SW) community is Ontology Based Data Access (OBDA), i.e., querying large volumes of assertional data through the vocabulary and semantics of ontologies. OBDA has recieved a lot of attention in the last years, however, while there have been advances on the theoretical side, e.g., the definition of OWL 2 QL, realizing efficient and scalable reasoning for large ontologies and large data sets is still problematic.

In this context, the most widespread reasoning technique for query answering is the *materialization* of inferences using *forward chaining*. This technique has several advantages, e.g., all inferences are done *off-line*, it is relatively easy to implement, and it offers high-performance at query time. However, if the terminological part of the ontology is large, materialization may require a long time and may considerably increase the storage requirements of the application. These disadvantages can turn this technique undesirable or unfeasible in several relevant use cases. An alternative to materialization is query answering by query rewriting, in which all reasoning is done *on-line*. Query rewriting has often been promoted as the most efficient way to query large volumes of data. However, in practice we have not seen a widespread adoption of these techniques. The main reason is that the queries generated by rewriting are often too large or too complex; for example, in the case of large OWL 2 QL ontologies, rewritings often generate hundreds or thousands of subqueries.

In this paper we present a technique that combines off-line and on-line reasoning to avoid the aforementioned issues of materialization and query rewriting and guaranteeing, in practice, minimal time and space for the construction of the triple store and fast query answering. We formulate the technique using RDBMS systems as the data backend; however, we note that the technique can easily be adapted to native triple stores. Likewise, in the following we focus on the OWL 2 QL direct entailment regime, however, the technique can also be used with the RDFS regime.

Semantic Index. The core idea of the semantic index technique is to encode the entailed hierarchies of the terminology of the OWL 2 QL ontology, i.e., the TBox $\mathcal{T}$, into *numeric indexes* that we assign to classes and properties. We use these values to insert the assertional data of the ontology, i.e., the ABox $\mathcal{A}$, into the DB, and use *range* queries to retrieve the triples entailed by the hierarchies and the ABox assertions. This allows us to create triple repositories that are almost the size of the original data and already encode most of the semantics of the ontology. Combined with a simple rewriting technique, we are able to provide fast and scalable query answering for SPARQL 1.1 *ABox queries* under the OWL 2 QL entailment regime while preserving soundness and completeness. Our proposal is strongly related to techniques for managing large transitive relations in

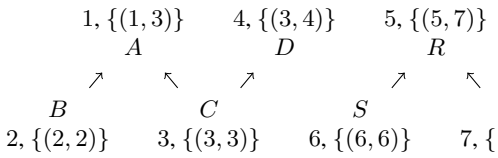

Fig. 1. D_C, D_R and the values for *idx* (left value) and *range* (in brackets) for Example 2. Arrows indicate sub-class or sub-property relations.

knowledge bases [2], however, our interest is not in the hierarchies themselves, as in [2], but in querying implicit assertional data that is associated to those hierarchies. Formally, a semantic index is defined as follows (we use DL notation due to space constraints).

Definition 1. *Given an OWL 2 QL TBox $\mathcal{T}$ and its vocabulary V of classes and properties,* a semantic index for $\mathcal{T}$ *is a pair of mappings $\langle idx, range\rangle$ with $idx : V \to \mathbb{N}$ and $range : V \to 2^{\mathbb{N}\times\mathbb{N}}$, such that, for each pair E_1, E_2 of classes or properties in V, we have that $\mathcal{T} \models E_1 \sqsubseteq E_2$ iff there is a pair $\langle \ell, h\rangle \in range(E_2)$ such that $\ell \leq idx(E_1) \leq h$.*

Given an OWL 2 QL ontology, we use the entailed class and property hierarchies to create $\langle idx, range\rangle$. Specifically, let $\mathcal{T}$ be the TBox, and D_C a minimal Directed Acyclic Graph (DAG) that represents the *entailed sub-class* relation between all named classes of $\mathcal{T}$ (i.e., the *transitive reduct* of the class hierarchy).[1] Then we can construct idx by initializing a counter $i = 0$, and visiting the nodes in D_C in a depth-first fashion starting from the root nodes. At each step and given the node N visited at that step, if $idx(N)$ is undefined, set $idx(N) = i$ and $i = i + 1$, else if $idx(N)$ is defined, backtrack until the next node for which idx is undefined. Now, to generate $range$, we visit the nodes in D_C starting from the leafs and going up. For each node N in the visit, if N is a leaf in D_C, then we set $range(N) = \{\langle idx(N), idx(N)\rangle\}$, and if N is not a leaf, then we set $range(N) = merge(\{\langle idx(N), idx(N)\rangle\} \cup \bigcup_{N_i \mid N_i \to N \in D_C} range(N_i))$, where $merge$ is a function that, given a set r of ranges, returns the minimal set r' of ranges that has coverage equal to r, e.g., $merge(\{\langle 5, 7\rangle, \langle 3, 5\rangle, \langle 9, 10\rangle\}) = \{\langle 3, 7\rangle, \langle 9, 10\rangle\}$. We proceed exactly in the same way with the DAG D_R representing the property hierarchy.

Example 2. Let A, B, C, D be classes, let R, S, M be properties, and consider the TBox $\mathcal{T} = \{B \sqsubseteq A, C \sqsubseteq A, C \sqsubseteq D, \exists R \sqsubseteq D, S \sqsubseteq R, M \sqsubseteq R\}$. Let the DAGs D_C and D_R for $\mathcal{T}$ be the ones depicted in Fig. 1, then the, technique will create idx and $range$ as indicated in the same figure. ■

We use a semantic index $\langle idx, range\rangle$ to insert the ABox in a DB as follows. We define the DB schema $\mathbf{R}$ with a table $T_C[\texttt{c1}, \texttt{idx}]$ for storing `rdf:type` assertions, and a table $T_R[\texttt{c1}, \texttt{c2}, \texttt{idx}]$ for storing property assertions, s.t. the columns `c1` and `c2` have type *uri* and `idx` has type $numeric$. Given an ABox $\mathcal{A}$, we insert the data in the DB such that for each $A(c) \in \mathcal{A}$ (i.e., triples of the form `c rdf:type A.`) we have the tuple $\langle c, idx(A)\rangle$ in T_C, and for each $P(c, c') \in \mathcal{A}$ (i.e., triples of the form `c P c'.`) we have $\langle c, c', idx(P)\rangle$ in T_R. The schema and the index allow us to

[1] We assume w.l.o.g. that $\mathcal{T}$ does not contain a cyclic chain of sub-class or sub-property axioms.

$$\sigma_{1\leq idx\leq 3}(T_C) \rightsquigarrow A(c_1) \qquad \sigma_{5\leq idx\leq 7}(T_R) \rightsquigarrow R(c_1, c_2)$$
$$\sigma_{2\leq idx\leq 2}(T_C) \rightsquigarrow B(c_1) \qquad \sigma_{6\leq idx\leq 6}(T_R) \rightsquigarrow S(c_1, c_2)$$
$$\sigma_{3\leq idx\leq 3}(T_C) \rightsquigarrow C(c_1) \qquad \sigma_{7\leq idx\leq 7}(T_R) \rightsquigarrow M(c_1, c_2)$$
$$\sigma_{3\leq idx\leq 4}(T_C) \rightsquigarrow D(c_1) \qquad \sigma_{5\leq idx\leq 7}(T_R) \rightsquigarrow D(c_1)$$

Fig. 2. The mappings for Example 2 created by our technique.

define, for each class A and each property P, a set of range queries over the DB that retrieve most URI's c, c' such that $\mathcal{O} \models A(c)$ or $\mathcal{O} \models P(c, c')$. E.g., if $range(A) = \{\langle 2, 35\rangle\}$, we use `'SELECT c1 FROM` T_C `WHERE idx >= 2 AND idx <= 35'`. We use these queries and all the entailments of $\mathcal{T}$ to define the DB mappings[2] of the system as follows:[3] *(i)* for each class A and each $\langle \ell, h\rangle \in range(A)$, we add the mapping $\sigma_{\ell\leq idx\leq h}(T_C) \rightsquigarrow A(c_1)$; *(ii)* for each property P and each $\langle \ell, h\rangle \in range(P)$, we add the mapping $\sigma_{\ell\leq idx\leq h}(T_R) \rightsquigarrow P(c_1, c_2)$; *(iii)* for each pair of properties P, P' such that $\mathcal{T} \models P'^{-} \sqsubseteq P$ and each $\langle \ell, h\rangle \in range(P')$ we add the mapping $\sigma_{\ell\leq idx\leq h}(T_R) \rightsquigarrow P(c_2, c_1)$; *(iv)* for each class A and each property P s.t. $\mathcal{T} \models \exists P \sqsubseteq A$ (resp., $\exists P^{-} \sqsubseteq A$) and each $\langle \ell, h\rangle \in range(P)$, we add the mapping $\sigma_{\ell\leq idx\leq h}(T_R) \rightsquigarrow A(c_1)$ (resp., $\sigma_{\ell\leq idx\leq h}(T_R) \rightsquigarrow A(c_2)$); *(v)* to eliminate redundancy, we replace any pair of mappings $\sigma_{\ell\leq idx\leq h}(T_C) \rightsquigarrow A(c_1)$ and $\sigma_{\ell'\leq idx\leq h'}(T_C) \rightsquigarrow A(c_1)$ such that $\ell' \leq h$ and $\ell \leq h'$ by the mapping $\sigma_{\min(\ell,\ell')\leq idx\leq \max(h,h')}(T_C) \rightsquigarrow A(c_1)$ (similarly for property mappings).

Example 3. For $\mathcal{T}$ and $\langle idx, range\rangle$ as in Example 2, we get the mappings in Fig. 2. ■

The *on-line* step of the technique is the SPARQL to SQL rewriting. This step is performed using a traditional SPARQL-algebra to relational-algebra translation, with the restriction that all the triples in the query graph must be translated to SQL views defined with the previous mappings. That is, given a query triple `?x rdf:type :A`, we translate it to the SQL union of all the queries in the mappings for `A`. Likewise for properties, given a query triple `?x :P ?y`, we translate it to the SQL union of the mappings for `P`. Since the specification of SPARQL 1.1 under OWL 2 QL entailments does not allow for true existentially quantified variables, this simple rewriting technique is sufficient for sound and complete query answering with SPARQL ABox queries.

Performance. We now provide a brief comparison of the performance of the semantic index vs. other reasoning techniques. This is only part of a larger evaluation that can be found online.[4] The evaluation is based on data from a real SW application, the *Resource Index* (RI) [3], winner of the Semantic Web Challenge Open Track 2010. The RI offers semantic search over 22 collections of biomedical documents. The semantics of the search is defined by the hierarchies of ≈ 200 ontologies that include well known and very large bio-medical ontologies like the Gene Ontology, SNOMEDCT, NCI Thesaurus, etc. The workflow of the RI can be summarized as follows: *(i)* using natural language processing, the RI annotates each document using the vocabulary of the ontologies, e.g., ABox triples of the form *Cervical_Cancer*(`':doc-224'`); *(ii)* the system expands the annotations using the ontologies and forward-chaining; *(iii)* users retrieve documents through queries, e.g,. `SELECT ?x WHERE {?x a :Cancer}`.

[2] DB mappings state how to retrieve the assertions for each class and property from the DB and are used during SPARQL to SQL rewriting. We refer to [5] for a formal description of these.

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

[4] https://babbage.inf.unibz.it/trac/obdapublic/wiki/resourceindex

For our tests we used one document collection, the *Clinical Trials.gov* (CT) documents, for which the annotation stage generates 181 million annotations, amounting to 14GB of data. In this case, a materialization of entailments using *forward-chaining* requires 7 days and 140GB of additional space. If the process is optimized (using data partitioning, parallelization, etc.) the time can be reduced to 40 minutes. In contrast, the offline stage of the semantic index only requires 27s and 4GB of additional space. To measure query answering performance, we issued several queries. Here we describe `'SELECT ?x WHERE {?x a DNA_Repair_Gene; a Antigen_Gene; a Cancer_Gene}'`. The query involves reasoning among 3 wide and deep hierarchies. It also has high selectivity, i.e., it only returns 2 distinct documents. In this case, if no rewriting is done and only forward-chaining is applied, the query executes in 3s (0.052s if the DB is warm). If we use rewritings of the form produced by QuOnto [1], Requiem [4], or Owlgres [7], these generate 467874 SQL queries, which the DB is not able to execute. If more succinct rewritings are used, e.g., those used by Presto [6], the rewriting is one SQL query with 274 comparisons (7,052B long), which requires 4.26s to execute (0.71s if warm). Using the semantic index technique, the SQL rewriting is one SQL query with 6 comparisons (357B long), which requires 3.58s to execute (0.08s if warm). For these tests we used DB2 v9.7 on a Linux system with a 2.67Ghz Intel Xeon CPU and 4GB of RAM.

These results show that the semantic index is an important improvement for semantic query answering. It outperforms all existing query rewriting techniques, offering the same level of performance as forward chaining at only a fraction of the storage cost. Future directions include extending our system[5] to allow for full SPARQL 1.1 and to apply the semantic index to reduce the cost of forward-chaining based reasoning in more expressive ontology languages.

Acknowledgements. We thank Dr. Paea LePendu for providing us with the RI data and supporting us in understanding this application. We thank Sergejs Pugacs for his work on the implementation and evaluation of the semantic index.

References

1. A. Acciarri, D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, and R. Rosati. QUONTO: QUerying ONTOlogies. In *Proc. of AAAI 2005*, pages 1670–1671, 2005.
2. R. Agrawal, A. Borgida, and H. V. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. In *Proc. of ACM SIGMOD*, pages 253–262, 1989.
3. P. LePendu, N. Noy, C. Jonquet, P. Alexander, N. Shah, and M. Musen. Optimize first, buy later: Analyzing metrics to ramp-up very large knowledge bases. In *Proc. of ISWC 2010*, volume 6496 of *LNCS*, pages 486–501. Springer, 2010.
4. H. Pérez-Urbina, B. Motik, and I. Horrocks. Tractable query answering and rewriting under description logic constraints. *J. of Applied Logic*, 8(2):186–209, 2010.
5. M. Rodríguez-Muro and D. Calvanese. Dependencies to optimize ontology based data access. In *Proc. of DL 2011*, volume 745 of *CEUR,* `ceur-ws.org`, 2011.
6. R. Rosati and A. Almatelli. Improving query answering over *DL-Lite* ontologies. In *Proc. of KR 2010*, pages 290–300, 2010.
7. M. Stocker and M. Smith. Owlgres: A scalable OWL reasoner. In *Proc. of OWLED 2008*, volume 432 of *CEUR,* `ceur-ws.org`, 2008.

[5] http://obda.inf.unibz.it/protege-plugin/