Practical RDF Schema Reasoning with Annotated Semantic Web Data

Carlos Viegas Damásio and Filipe Ferreira

CENTRIA, Departamento de Informática Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
cd@di.fct.unl.pt, p110362@fct.unl.pt

Abstract. Semantic Web data with annotations is becoming available, being YAGO knowledge base a prominent example. In this paper we present an approach to perform the closure of large RDF Schema annotated semantic web data using standard database technology. In particular, we exploit several alternatives to address the problem of computing transitive closure with real fuzzy semantic data extracted from YAGO in the PostgreSQL database management system. We benchmark the several alternatives and compare to classical RDF Schema reasoning, providing the first implementation of annotated RDF schema in persistent storage.

Keywords: Annotated Semantic Web Data, Fuzzy RDF Schema, Transitive closure in SQL, Rules.

1 Introduction

The Semantic Web rests on large amounts of data expressed in the form of RDF triples. The need to extend this data with meta-information like trust, provenance and confidence [26,22,3] imposed new requirements and extensions to the Resource Description Framework (Schema) [19] to handle annotations appropriately. Briefly, an annotation v from a suitable mathematical structure is added to the ordinary triples $(s\ p\ o)$ obtaining $(s\ p\ o) : v$, annotating with v the statement that subject s is related via property p to object o. The general semantics of this RDFS extension has been recently addressed [21,22] improving the initial work of [26], but only a memory-based Prolog implementation is available.

The feasibility of large scale classical RDFS reasoning and its extensions has been shown in the literature [27,10,25,13,11]. In this paper we will show that the inclusion of annotated reasoning naturally introduces some overhead but that it is still possible to perform the closure of large annotated RDFS data in reasonable amount of time. The major difficulty is the implementation of transitive closure of RDF Schema `subPropertyOf` and `subClassOf` properties, with the extra problem of maintaining the annotations since a careless implementation might result in worst-case exponential runtime. For this reason, we discuss several alternative approaches for implementing transitive closure of RDFS data exploiting facilities present in modern relational database systems, particularly recursive views. We selected PostgreSQL[1] because of the mechanisms it provides as well as ease of integration with other Semantic Web triple stores and reasoners, and use annotated data of the YAGO ontology [24] to evaluate our contribution. The use of existing semantic web data is essential since current "artificial" benchmarks like LUBM [7] do not reflect exactly the patterns of data present in real applications [6]. All the data and code is made publicly available at `http://ardfsql.blogspot.com/`, including instructions for replicating the tests presented in the paper.

An advantage of our approach is that we present an entirely based SQL implementation allowing practitioners to use our technique directly in their preferred standard RDBMS without external imperative code. To the best of our knowledge, we present the first complete database implementation of the fuzzy Annotated Resource Description Framework Schema and assess it with respect to real ontologies. We justify that our approach is competitive and show that semi-naive and a variant of semi-naive (differential) evaluation are not always competitive for performing RDFS closures, especially when annotated data is present.

In the next section, we start by overviewing the Annotated RDF Schema framework. In Section 3 we address the issues storing (annotated) RDFS data in persistent storage, and present our own encoding. Section 4 describes the several closure algorithms for RDFS with annotated data whose benchmarking is performed in Section 5. We proceed with comparisons to relevant work in Section 6 and we finish the paper in Section 7 with conclusions and future work.

2 Annotated RDF Schema

In this section we shortly present the RDFS with Annotations [22] and we assume good knowledge of the classical (or crisp) Resource Description Framework (RDF). We ignore the model-theoretical aspects and focus on the inference rules used to perform the closure. An annotation domain [22] is an algebraic structure $D = \langle L \preceq, \otimes, \top, \bot \rangle$ such that $\langle L, \preceq, \top, \bot \rangle$ is a bounded lattice (i.e. a lattice with a $\top$ top and $\bot$ bottom elements) and where operator $\otimes$ is a t-norm. A t-norm is a generalization of the conjunction operation to the many-valued case, obeying to the natural properties of commutativity, associativity, monotonicity and existence of a neutral element (i.e. $v \otimes \top = \top \otimes v = v$).

The inference rules for annotated RDFS [22] can be found in Fig. 1 and where `sp`, `sc`, `type`, `dom` and `range` are abbreviations for the RDF and RDFS properties `rdfs:subPropertyOf`, `rdfs:subClassOf`, `rdf:type`, `rdfs:domain` and `rdfs:range`, respectively, which is known as the ρdf vocabulary or minimal RDFS [16]. The rules are extensions of the set of crisp rules defined in [16] to handle annotations; the original rules can be obtained by dropping the annotations or equivalently using the algebraic domain $D_{01} = \langle \{0,1\}, \leq, \min, 0, 1 \rangle$, which corresponds to classical boolean logic. As standard practice, we drop reflexivity rules which provide uninteresting inferences like that any class (property) is a

[1] See `http://www.postgresql.org/`

1. Subproperty

$(a)\ \frac{(A,\texttt{sp},B):v_1,(B,\texttt{sp},C):v_2}{(A,\texttt{sp},C):v_1 \otimes v_2}$

$(b)\ \frac{(A,\texttt{sp},B):v_1,(X,A,Y):v_2}{(X,B,Y):v_1 \otimes v_2}$

2. Subclass

$(a)\ \frac{(A,\texttt{sc},B):v_1,(B,\texttt{sc},C):v_2}{(A,\texttt{sc},C):v_1 \otimes v_2}$

$(b)\ \frac{(A,\texttt{sc},B):v_1,(X,\texttt{type},A):v_2}{(X,\texttt{type},B):v_1 \otimes v_2}$

3. Typing

$(a)\ \frac{(A,\texttt{dom},B):v_1,(X,A,Y):v_2}{(X,\texttt{type},B):v_1 \otimes v_2}$

$(b)\ \frac{(A,\texttt{range},B):v_1,(X,A,Y):v_2}{(Y,\texttt{type},B):v_1 \otimes v_2}$

4. Implicit Typing

$(a)\ \frac{(A,\texttt{dom},B):v_1,(C,\texttt{sp},A):v_2,(X,C,Y):v_3}{(X,\texttt{type},B):v_1 \otimes v_2 \otimes v_3}$

$(b)\ \frac{(A,\texttt{range},B):v_1,(C,\texttt{sp},A):v_2,(X,C,Y):v_3}{(Y,\texttt{type},B):v_1 \otimes v_2 \otimes v_3}$

5. Generalization

$\frac{(X,A,Y):v_1,(X,A,Y):v_2}{(X,A,Y):v_1 \vee v_2}$

Fig. 1. Inference rules for annotated RDFS [22]

subclass (subproperty) of itself [16]. An important aspect of the inference rules is that, with the exception of rules *1b)* and *5)*, the conclusions of the rules produce triples of the ρdf vocabulary.

In this paper we will restrict mostly to the annotation domain $D_{goedel} = \langle [0,1], \leq, \min, 0, 1 \rangle$ where the t-norm is the minimum operator and the least upper bound is the maximum operator corresponding to Goedel's fuzzy logic [8], obtaining the fuzzy RDFS framework in [21]. Our algorithms will work and terminate with any t-norm in the $[0,1]$ interval. Termination is guaranteed in these circumstances by our own results on fuzzy logic programming [5], and are far from trivial since an infinitely-valued lattice is being used. In fact, one of the distinctions between the work [26] and [22], is that the former is restricted to finite annotation lattices while the latter allows infinite ones. A particularly striking example of the problems that can occur is the use of real-valued product t-norm which can generate "new" values from existing ones, contrasting to the minimum t-norm that can only return existing annotations in the asserted data.

3 Persistent Storing of Annotated RDFS Data

We follow a hybrid approach for representing RDFS data in a relational database with some optimizations. We use a vertically partitioned approach [1] for storing triples (in fact quads) having a table for each of the properties of the ρdf vocabulary, and use a common table to store all the other triples (see Fig. 2). The major differences to a classical relational RDFS representation is that we add an extra column to represent the double valued annotations, and include a column to store graph information in order to be compatible with SPARQL. The schema used is very similar to the one of Sesame's reported in [4] but we do not use property-tables. The `id` column is the key of `Triples` table automatically filled-in with a sequence, and `ref` column has been added to allow traceability support to the reification vocabulary of RDF Schema but it is not used in the current implementation. Moreover, we use a `Resources` table to reduce the size of the database by keeping a single entry for each plain literal (`nodet=1, value="`*`string`*`", type="`*`language tag`*`" (or NULL)`), typed literal (`nodet=2, value="`*`string`*`",type="`*`URI type`*`"`), URI (`nodet=3,value="`*`URI`*`",type=NULL`)

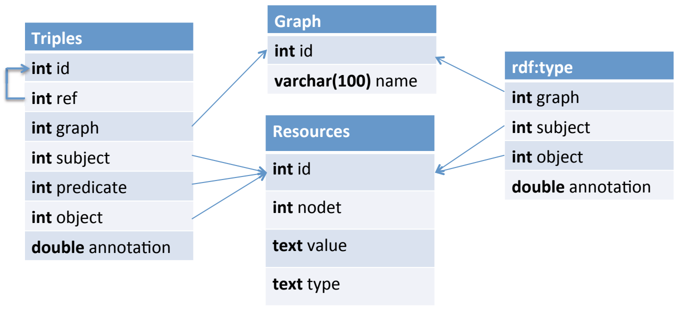

Fig. 2. Annotated RDFS table schema

or blank node (`nodet=4`, `type="`*`identifier`*`"`, `type=NULL`) in any asserted data. The `Resources` table is pre-initialized at database creation time with the ρdf vocabulary in order to have known identifiers for these URIs, reducing joins. Figure 2 shows only the table for `rdf:type`, being the schema of the remaining tables for the properties of the ρdf vocabulary identical. In the common table `Triples`, and ρdf tables `rdf:type`, `rdfs:subPropertyOf`, `rdfs:subClassOf`, `rdfs:domain` and `rdfs:range` we only have an integer foreign key for the subject, property and object of each triple, making each row fixed-length.

Notice that this representation is very similar to the star schema used in data warehouses where fact tables correspond to our triple and property specific tables while dimension tables correspond to our resources table. Moreover as discussed in [1], this representation has several advantages for query answering because it reduces self-joins in tables, which will benefit from when implementing the inference rules presented in Section 3. The choice of a particular schema is not arbitrary and can have significative impact in performance (see [1]). Additionally, we realized that under default database configuration and with the tested data sets, indexing did not bring significant advantages for performing the annotated RDFS closure for the data available, and therefore we do not include any on-disk indexes.

4 Closure of Annotated RDFS Data

In this section we discuss the techniques and algorithms we have developed to perform the closure of Annotated RDFS data. We start by discussing the annotated RDFS specific generalization rule, and afterwards we discuss rule order application. We conclude that the only recursive rules necessary are subproperty and subclass transitivity (rules 1a and 2a), which can be implemented in any current mainstream RDBMSs. The major difficulty is performing the transitive closure with annotated data, for which we specify several algorithms.

4.1 Generalization Rule

The rule which has greater impact in the implementation is the generalization rule which simply states whenever a triple is derived twice with different annotations $(s\ p\ o) : v_1$ and $(s\ p\ o) : v_2$ then their annotations may be combined to obtain a triple with a greater annotation $(s\ p\ o) : v_1 \vee v_2$, and the original annotated triples can be removed. In the case of the $[0, 1]$ interval ordered as usual, this rule corresponds to keeping the annotated triple with maximum value, deleting any other smaller annotated triple. It is important to realize that if subsumed annotated triples are left in the database, then exponential behaviour can be generated for some t-norms.

Example 1. Consider the following annotated RDFS database in the algebraic domain $D_{prod} = \langle [0, 1], \leq, \times, 0, 1\rangle$ where the t-norm is the usual real-valued multiplication.

$$
\begin{array}{lll}
(a_0, \mathtt{sc}, b_0) : \frac{3}{1000} & (a_1, \mathtt{sc}, b_1) : \frac{7}{1000} & (a_2, \mathtt{sc}, b_2) : \frac{13}{1000} \\
(a_0, \mathtt{sc}, c_0) : \frac{5}{1000} & (a_1, \mathtt{sc}, c_1) : \frac{11}{1000} & (a_2, \mathtt{sc}, c_2) : \frac{17}{1000} \\
(b_0, \mathtt{sc}, a_1) : 1 & (b_1, \mathtt{sc}, a_2) : 1 & (b_2, \mathtt{sc}, a_3) : 1 \\
(c_0, \mathtt{sc}, a_1) : 1 & (c_1, \mathtt{sc}, a_2) : 1 & (c_2, \mathtt{sc}, a_3) : 1
\end{array}
$$

There are 8 paths between a_0 and a_3 by selecting at each step a path via b_i or a c_i, having assigned a different annotation obtained by multiplying the annotations in the path edges going out of each a_i, originating 8 subclass annotated triples $(a_0, \mathtt{sc}, a_3)$ each with a different annotation. It is immediate to see that the construction can be iterated more times with different prime-number based annotations, obtaining an exponential number of subclass relations on the number of a nodes, all with different annotations. By applying the generalization rule one can see that all but one of these annotated triples are redundant.

For this reason, our major concern will always be to never introduce in the tables duplicated triples with different annotations. To achieve this we will extensively resort to a mixture of SQL aggregations using `MAX` function, and in-built the rule in the other rules. Therefore, we will only take care of domains over the real-valued $[0, 1]$ to achieve better performance, otherwise one would require from the DBMS facilities to implement new aggregation functions[2]. The encoding of inference rule $3a$ in SQL can be seen below, where `i_graph` is a parameter with the graph identifier to be closed:

```
CREATE OR REPLACE FUNCTION Rule3a(i_graph integer) RETURNS integer AS $typa$
BEGIN

 UPDATE "rdf:type" as r SET annotation=q.annotation FROM
    (SELECT t.graph, t.subject, d.object, MAX(tnorm(d.annotation,t.annotation)) AS annotation
     FROM "rdfs:domain" d INNER JOIN "Triples" t ON (d.subject=t.predicate)
     WHERE d.graph=i_graph AND t.graph=i_graph
     GROUP BY t.subject, d.object, t.graph ) AS q
    WHERE (r.subject,r.object,r.graph)=(q.subject,q.object,q.graph) AND
           r.annotation<q.annotation;

 INSERT INTO "rdf:type" (
     SELECT t.graph, t.subject, d.object, MAX(tnorm(d.annotation,t.annotation)) AS annotation
     FROM "rdfs:domain" d INNER JOIN "Triples" t ON (d.subject=t.predicate)
     WHERE d.graph=i_graph AND t.graph=i_graph AND
```

[2] These are available in some commercial RDBMSs.

```
        NOT EXISTS (SELECT * FROM "rdf:type" AS old WHERE old.graph=i_graph
                    AND old.subject=t.subject AND old.object=d.object)
     GROUP BY t.subject, d.object, t.graph );

 RETURN 1;
END
```

First we update the table `rdf:type` table with the better inferred annotations for already existing triples and afterwards we `INSERT` completely new triples accordingly to the `NOT EXISTS` clause. Both statements only generate an annotation for a given triple therefore not introducing redundant information. The user-defined function `tnorm` implements in a stored function the intended t-norm function (in our experiments, minimum).

4.2 Fixpoint Iteration and Rule Ordering

The rules present in Fig. 1 have to be iterated till a fixpoint is reached, i.e. no new annotated triples are generated. A clever implementation does not require the execution of the whole set of rules at a time. It is easy to see that rules depend on each other (see Fig. 3) and rules can be ordered to reduce computation time.

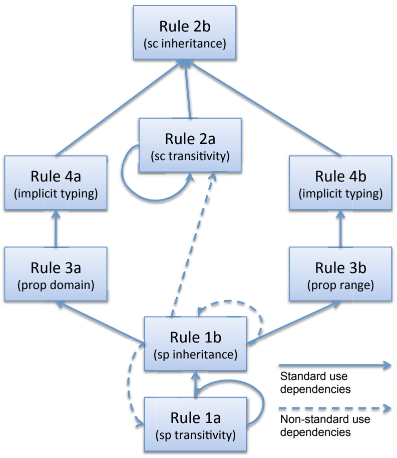

Fig. 3. Dependency graph of annotated RDFS inference rules

The dependency graph of Fig. 3 is very similar to the one presented in [27] with three distinctive features. First, we have the implicit typing rules, and we show dependencies regarding "non-standard" use of ρdf vocabulary (the dashed dependencies). More important is that we do not have to iterate the subclass and subproperty inheritance rules (rules 1b and 2b) because of the following result:

Lemma 1. *Subclass inheritance rule is idempotent*[3], *and subproperty inheritance is idempotent with standard use of vocabulary.*

Proof. Consider two chained applications of rule 2*b*

$$1.\ \frac{(A,\texttt{sc},B):v_1,(X,\texttt{type},A):v_2}{(X,\texttt{type},B):v_1\otimes v_2}\qquad 2.\ \frac{(B,\texttt{sc},C):v_3,(X,\texttt{type},B):v_1\otimes v_2}{(X,\texttt{type},C):v_1\otimes v_2\otimes v_3}$$

However since the subclass relationship is closed with respect to rule 2*a* we conclude that $(A,\texttt{sc},C):v_1\otimes v_3$ and thus we will get by one application of rule 2*b* that $(X,\texttt{type},C):v_1\otimes v_2\otimes v_3$, by commutativity and associativity of $\otimes$:

$$\frac{(A,\texttt{sc},B):v_1,(B,\texttt{sc},C):v_3}{(A,\texttt{sc},C):v_1\otimes v_3}\qquad 3.\ \frac{(A,\texttt{sc},C):v_1\otimes v_3,(X,\texttt{type},A):v_2}{(X,\texttt{type},B):v_1\otimes v_3\otimes v_2}$$

If in the subclass transitivity closure we get an annotation for $(A,\texttt{sc},C):v_4$ such that $v_4>v_1\otimes v_3$ the inferred triple by chained application in step 2 will be subsumed by the triple in corresponding step 3.

A similar argument shows the result for the case of subproperty inheritance, but in this situation we have to guarantee that we cannot generate triples where $B=\texttt{sp}$. However, this can only happens whenever we have a statement $(A,\texttt{sp},\texttt{sp}):v$ in the original graph, i.e. with non-standard use of the ρdf vocabulary.

We believe that it has not been realized before in the literature that a single application of rules 1*b* and 2*b* is necessary to generate all the triples, whenever the subclass and subproperty relationships are closed, even though apparently it is assumed to hold in [9] for the classical case. Therefore, we conclude that the only recursive rules necessary to perform annotated RDFS closure are transitive closures. This analysis carries over for the classical RDFS case as well.

Assuming standard use of the ρdf vocabulary we are guaranteed that rule 1*b* inserts data only in table `Triples`, further simplifying its implementation.

4.3 Transitive Closure with Annotated Data

In order to conclude the implementation of annotated RDFS we just have to consider transitivity of the subclass and subproperty relations. The definition of transitive closure and shortest-path algorithms for databases has been addressed in the literature [14,2,18,15].

To formalize our algorithms we will resort to an extension of the notion of fuzzy binary relation [28]. A fuzzy binary relation R is a mapping $R:U\times U\rightarrow[0,1]$, associating to each pair of elements of the universe U a membership degree in $[0,1]$. This can be appropriately generalized to our setting by defining annotated relations over annotated domain $D=\langle L,\preceq,\otimes,\top,\bot\rangle$ as mappings $R:U\times U\rightarrow L$.

[3] By idempotent rule we mean that the rule will always produce the same results whether it is applied once or several times, thus there is no need to apply it more than once.

Definition 1. *Consider binary annotated relations R_1 and R_2 with universe of discourse $U \times U$ and over annotated domain $D = \langle L \preceq, \otimes, \top, \bot\rangle$. Define composition $R_1 \circ R_2$ and union $R_1 \vee R_2$ of annotated relations R_1 and R_2 as:*

$$(R_1 \circ R_2)(u,w) = \bigvee_{v \in U} \{R_1(u,v) \otimes R_2(v,w)\} \quad (R_1 \vee R_2)(u,v) = R_1(u,v) \vee R_2(u,v)$$

In the case of fuzzy annotated domains of the form $D = \langle [0,1], \leq, \otimes, 1, 0\rangle$, we can use relational algebra to implement the composition and union of annotated relations. For ease of presentation we overload the symbols of $\vee$ and $\circ$:

Definition 2. *Consider binary fuzzy annotated relations R_1 and R_2, represented in relational algebra by relations $r_1 \subseteq U \times U \times [0,1]$ and $r_2 \subseteq U \times U \times [0,1]$ with relational schema (sub, obj, ann) and obeying to functional dependency $sub\ obj \rightarrow ann$. Operations $r_1 \circ r_2$ and $r_1 \vee r_2$ are defined as:*

$$\begin{aligned} r_1 \circ r_2 = {}& {}_{sub,obj}\mathcal{G}_{MAX(ann)} (\\ & \Pi_{r1.sub \text{ as } sub,\ r2.obj \text{ as } obj,\ r1.ann \otimes r_2.ann \text{ as } ann}\ \sigma_{r1.obj = r2.sub}(r_1 \times r_2)) \\ r_1 \vee r_2 = {}& {}_{sub,obj}\mathcal{G}_{MAX(ann)}(r_1 \cup r_2) \end{aligned}$$

Using standard relational algebra notation, where Π is the projection operator, $\mathcal{G}$ is the aggregation operator, σ the selection operator and $\times$ the Cartesian-Product operation[4]*. Finally $\otimes$ is the t-norm operation.*

To simplify matters, we have assumed that only non-zero annotated information will be present in the relations, reducing a lot the storage requirements. Since $0 \vee v = v \vee 0 = v$ and $0 \otimes v = v \otimes 0 = 0$ we will be able to respect all the original operations in relational algebra.

The transitived closure of annotated fuzzy binary relations R^+ is defined by the least fixpoint of the equation $R^+ = R \vee (R^+ \circ R)$, corresponding to determining shortest-paths between all nodes in a weighted graph. In Fig. 4 we present five different ways of obtaining the annotated fuzzy relation R^+, corresponding to well-known transitive closure algorithms found in the literature [14,2,18,15]. The naive algorithm is a direct implementation of an iterated computation of the fixpoint of the recursive definition. The popular semi-naive algorithm is an improvement of the first algorithm, by at each step just propagating the changes performed in the previous iteration. The differential semi-naive algorithm is particularly optimized for relational database systems and was proposed and implemented in the DLV^{DB} system [25], which reduces the number of joins necessary with respect to the semi-naive algorithm. The matrix algorithm corresponds to Warshall's algorithm, and the logarithmic algorithm is a variant specially constructed for reducing joins [18]. We show below the translation to SQL of the Matrix method that will prove to be the best and more reliable algorithm:

[4] For the exact definition of these operators see any standard database manual, e.g. [20].

Naive algorithm

```
$R^+ = R$
LOOP
 $R^+ := R \vee (R^+ \circ R)$
WHILE $R^+$ changes
```

Semi-naive algorithm

```
$old_R := \emptyset$
$R^+ = R$
$\delta = R$
WHILE $\delta \neq \emptyset$
 $old_R := R^+$
 $R^+ := R^+ \vee \delta \circ R$
 $\delta := R^+ - old_R$
END
```

Differential semi-naive alg.

```
$R^+ = R$
$\delta = R$
LOOP
 $\Delta := (R^+ \circ \delta) \vee (\delta \circ R^+) \vee (\delta \circ \delta)$
 $\Delta := (\Delta - \delta) - R^+$
 $R^+ := R^+ \vee \delta$
 $\delta = \Delta$
WHILE $\delta \neq \emptyset$
```

Matrix algorithm

```
$R^+ = R$
LOOP
 $R^+ := R^+ \vee (R^+ \circ R^+)$
WHILE $R^+$ changes
```

Logarithmic algorithm

```
$R^+ = R$
$\Delta = R$
$\delta = R$
LOOP
 $\delta := \delta \circ \delta$
 $\Delta := R^+ \circ \delta$
 $R^+ := R^+ \vee \delta \vee \Delta$
WHILE $R^+$ changes
```

Fig. 4. Transitive closure algorithms for annotated binary relations

```
CREATE OR REPLACE FUNCTION MatrixRule2a(i_graph integer) RETURNS integer
DECLARE
   nrow_upd integer;
   nrow_ins integer;
BEGIN
 LOOP
  UPDATE "rdfs:subClassOf" as r SET annotation=a.annotation FROM (
  SELECT  q1.graph, q1.subject, q2.object, MAX(tnorm(q1.annotation,q2.annotation)) annotation
  FROM "rdfs:subClassOf" AS q1 INNER JOIN "rdfs:subClassOf" AS q2 ON ( q1.object=q2.subject )
  WHERE q1.graph=i_graph AND q2.graph=i_graph
  GROUP BY q1.subject, q2.object, q1.graph
 ) AS a
  WHERE (r.subject,r.object,r.graph)=(a.subject,a.object,a.graph) AND r.annotation<a.annotation;
  GET DIAGNOSTICS nrow_upd = ROW_COUNT;

 INSERT INTO "rdfs:subClassOf" (
  SELECT  q1.graph, q1.subject, q2.object, MAX(tnorm(q1.annotation,q2.annotation)) annotation
  FROM "rdfs:subClassOf" AS q1 INNER JOIN "rdfs:subClassOf" AS q2 ON ( q1.object=q2.subject )
  WHERE q1.graph=i_graph AND q2.graph=i_graph AND
  NOT EXISTS (SELECT * FROM "rdfs:subClassOf" AS sc
              WHERE sc.subject = q1.subject AND sc.object=q2.object AND sc.graph=q1.graph)
  GROUP BY q1.subject, q2.object, q1.graph );
  GET DIAGNOSTICS nrow_ins = ROW_COUNT;

    IF (nrow_upd+nrow_ins=0) THEN
     EXIT;
    END IF;
   END LOOP;

 RETURN 1;
END
```

As before the translation proceeds in two steps. First, we update the annotation of any already existing `rdfs:subClassOf` triple. Afterwards, we insert new `rdfs:subClassOf` triples of newly generated paths between nodes. In the case of the classical transitive closure implementation, it is not necessary to have the `UPDATE` statement, and that aggregation with `GROUP BY` to obtain the `MAX` of t-norm combined annotations, and of course the annotations. Therefore it is expected to have overhead with respect to the classical RDFS closure. We have implemented all the algorithms being the translation of the formal descriptions in Fig. 4 to SQL along the same lines.

5 Evaluation

In this section we present and discuss the results of the evaluation of the algorithms that we have developed. We start presenting the evaluation methods and the tests we have performed. First, we compare the several algorithms for classical RDFS closure and compare to the DLV^{DB} system [25]. We proceed by performing the testing with the annotated version algorithms in order to test effectiveness and obtain the overhead.

The algorithms were evaluated with respect to completeness and performance. We started by guaranteeing that the RDFS closure algorithms produced the correct outputs, namely that the output does not have duplicate or missing triples and, in the case of annotated graphs, that the annotations in each annotated triple are correct. For that matter, we produced several small tests (that we believe are representative of at least most kinds of graphs) to test the correctness of the implementation of each method. Afterwards we compared the number of triples in the output of each method to each other and with the DLV^{DB} system, for the same test. Since every method, and DLV^{DB}, returns the same amount of triples for each test, we have reasons to believe in the correctness of our implementation. The second aspect of evaluation was the amount of time that each method took to compute the closure for each test, which we now detail. The tests were built from the data sets YAGO, YAGO2 and WordNet 2.0. We chose YAGO and YAGO2 as our main sources of testing sets because YAGO contains large amounts of data, annotated with values in the [0,1] interval, so its the ideal dataset for the evaluation of annotated RDFS closure. Since WordNet was used in past works as test data set, in order to be possible to compare with other applications, we also tested the closure of this data set.

We devised six tests[5] to the Recursive, Semi-naive, Matrix-based, Differential, Logarithmic implementations and annotated versions. For the tests 1, 2, 3 and 4 only the code for the rule 2a) was executed since the objective of the tests is to show the way the diferent implementations react to transitive closure. For the remaining tests the full RDFS closure was computed. We also tested the closure time using DLV^{DB}. The description, sizes of the test sets and the output are shown in Table 1.

[5] The tested data and all the code needed to perform the tests is available at http://ardfsql.blogspot.com/

Table 1. Test sizes and specification

Test	1^*	2^*	3^*	4^*	5^+	6^+
Input Size	0.066M	0.366M	0.599M	3.617M	0.417M	1.942M
Output Size	0.599M	3.617M	0.599M	3.617M	3.790M	4.947M

* - only transitive closure of `rdfs:subClassOf`

$^+$ - full RDFS closure of the input data

Test 1: Contains `rdfs:subClassOf` data from YAGO in the WordNetLinks file.
Test 2: Contains all `rdfs:subClassOf` data from YAGO2.
Test 3: Contains the output graph from Test 1.
Test 4: Contains the output graph from Test 2.
Test 5: Full RDFS closure of a subset of YAGO, containing the subclass data of the WordNetLinks file, all `rdfs:subPropertyOf`, `rdfs:domain`, `rdfs:range` triples and the triples from relations created, givenNameOf, inTimeZone, isLeaderOf, isPartOf, isSubstanceOf.
Test 6: Full RDFS closure of the WordNet 2.0 Full data set.

The tests were performed using a Laptop with an Intel i5 2.27GHz processor, 4Gb of RAM and running Windows 7 64-bit. The PostgreSQL 9.0 database was installed with default options, and no modification was made to the DB server. The data for each test was stored in a new database with no use of commands for the gathering of statistics. All database constraints were disabled and no indexing was used. We repeated each test three times, and in this paper we present the reasoning average time, excluding as usual data loading time.

Table 2. Results for the classical algorithms, time results presented in seconds

	1		2		3		4		5		6	
	Time	Deviation	Time	Deviation	Time	Deviation	Time	Deviation	Time	Deviation	Time	Deviation
Matrix	32.33	0.45%	325.18	17.93%	**8.93**	11.00%	**69.85**	5.24%	85.49	5.24%	103.06	2.24%
Logarithmic	41.99	1.23%	283.01	6.03%	60.50	0.59%	244.30	1.68%	89.02	1.45%	84.78	19.64%
Differential	38.53	0.84%	507.70	43.84%	48.87	2.96%	402.00	5.81%	169.45	6.81%	99.07	25.00%
Semi-Naive	142.10	1.04%	936.91	2.49%	40.83	0.38%	253.07	5.97%	188.66	3.57%	**73.65**	5.72%
Recursive	**12.18**	0.62%	**87.99**	1.29%	*	*	*	*	**65.71**	2.35%	96.89	22.25%
DLVdb	39.13	1.55%	327.36	7.72%	20.62	1.93%	133.54	3.37%	161.94	2.35%	179.14	4.35%

* timeout

The results for the classical algorithms for the six tests can be found in Table 2, where the best results for each test are in bold. The recursive implementation, using built-in recursive views of PostgreSQL, is either very well-behaved or extremely bad. In general, the semi-naive method has poor performance when compared to the best algorithm in all tests, except for test 6. The differential semi-naive performs better than semi-naive in tests 1,2 and 5, where the graph closure contains a large number of new triples obtained by transitive closure of the `rdf:subClassOf` relation. The semi-naive and differential semi-naive methods have worse performance than the other algorithms, justifying also the comparatively bad timings of the DLV^{DB} system which uses the differential one. The logarithmic method can be very good but its behavior oscillates more than the matrix method although they have similar performances.

The results of test 6 are particularly significant since in [27] the same task is performed in a 32 cluster machine in more than 3 minutes. In test 6 the five methods implemented by us have very similar times, in fact the algorithms with worse performance so far (semi-naive and differential) seem to perform much better than in test 5. This happens because the difference in graph closure times between the implementations comes from the time taken evaluating the transitive closure rules (rules `1a` and `2a`) and in test 6 the number of triples in the `rdfs:subPropertyOf` and `rdfs:subClassOf` relations is not big enough (respectivly 11 and 42) to show efficiency differences among transitive closure algorithms.

We have also performed a partial test with larger data for the classical case for determining the full RDFS closure of a larger subset of YAGO, containing all `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:domain`, `rdfs:range` triples all `rdf:type` triples except those in "IsAExtractor", plus the triples of the YAGO relations bornOnDate, directed, familyNameOf, graduatedFrom, isMemberOf, isPartOf, isSubstanceOf, locatedIn, and worksAt. This test has 5.505M input triples generating 29.462M output triples. We have run the matrix and logarithmic methods obtaining 1547 seconds and 1947 seconds, respectively.

Table 3. Results for the annotated algorithms, time results presented in seconds

	1			2			3		
	Time	Deviation	Overhead	Time	Deviation	Overhead	Time	Deviation	Overhead
Matrix	**113.06**	3.52%	253%	1484.91	21.70%	356%	**30.48**	2.63%	241%
Logarithmic	126.90	1.27%	202%	**829.24**	10.24%	193%	155.02	0.39%	156%
Differential	163.36	4.86%	323%	1324.84	61.73%	160%	153.33	2.09%	213%
Semi-Naive	230.15	1.94%	61.96%	2761.80	115.54%	195%	149.05	3.05%	265%
Recursive	6679.70	57.74%	5496%	*	*	*	*	*	*

	4			5			6		
	Time	Deviation	Overhead	Time	Deviation	Overhead	Time	Deviation	Overhead
Matrix	**230.14**	0.01	229%	272.59	4.74%	218%	336.92	20.74%	226%
Logarithmic	1033.00	0.37	322%	**223.04**	3.71%	150%	341.63	35.34%	302%
Differential	978.63	0.10	143%	458.25	1.04%	170%	337.37	8.97%	240%
Semi-Naive	2177.60	1.16	760%	336.00	1.26%	78%	**195.18**	6.11%	165%
Recursive	*	*	*	6075.24	30.48%	914%	344.89	6.45%	245%

* timeout

The results for the annotated versions of the algorithm can be found in Table 3. It is clear that the recursive version does not scale with annotations, which we believe is due to the problem identified in Lemma 1. Most of the times the matrix algorithm is better and the logarithmic algorithm suffers from large variance problems. The differential algorithm is well-behaved in the case of the WordNet test data but most of the times it is not competitive.

The overhead introduced by the annotated versions is consistently around 250% for the case of the matrix method. This is expected since more queries and more complex are necessary to obtain the closure.

We have also performed some tests using a different database server configuration and indexes in specific tables. In these tests annotated and non annotated versions of the algorithms have performed better than with default configuration, in some cases three or four times faster. This leads us to believe that server configuration optimization may lead to great improvement of performance of these algorithms. This was expected and future work will be developed in order to provide more definitive conclusions.

6 Comparison and Related Work

Current triple stores like Sesame[6] and Jena[7] apparently do not perform the RDFS closure directly in the RDBMSs over stored data, and first load the data into memory, perform the inferences and store them afterwards. Moreover, their inference reasoning algorithms do not handle annotated data.

An extensive analysis of inferencing with RDF(S) and OWL data can be found in the description of the SOAR system [10]. A first major distinction to our approach is that part of the information is kept in main-memory, basically to what corresponds to our special tables for handling ρdf vocabulary and thus the transitive closure is performed in main memory, and no annotations are available. However, SOAR has more rules and implements a subset of OWL inferences, which we do not address here. It is used a technique call partial-indexing which relies on pre-processing a comparatively small-sized T-box with respect to the assertional data [11]. This terminological data extends the knowledge in our `rdfs:subClassof`, `rdfs:subPropertyOf`, `rdfs:domain`, and `rdfs:range` properties, and is not the major concern of the authors.

A novel extension of the SOAR system has been reported with scalable and distributed algorithms to handle particular annotations for dealing with trust/provenance including blacklisting, authoritativeness and ranking [3]. They cover a subset of the OWL 2 RL/RDF rules in order to guarantee a linear number of inferences on the size of the assertional data. The computation of the transitive closure of subclass and subproperty relations with annotations are performed by semi-naive evaluation with specialized algorithms, and thus it is not comparable to our approach.

Here we are not trying to assess query and storage trade-offs like [23] where just part of the RDFS closure graph is stored and the remaining triples are inferred at query time. However, we believe the same kind of balance still holds for annotated RDFS Semantic Web data.

Our approach also differs from [12] since we do not allow neither SPARQL querying nor change of entailment regime. A quite recent improvement of the annotated Semantic Web framework has been made available at [17,29]. The semantics have been extended with aggregate operators, and AnQL an extension of SPARQL to handle annotations has also been detailed with an available

[6] See `http://www.openrdf.org/`

[7] See `http://jena.sourceforge.net/`

memory-based implementation in Prolog using constraint logic programming. Since persistent storage is not supported we did not compare with this approach.

7 Conclusions and Future Work

In this paper we present a full relational database implementation of the annotated RDFS closure rules of [22] supporting any t-norm real-valued intervals, namely in the unit interval $[0, 1]$. We have analysed the dependencies of inference rules and proved that recursion is solely necessary for performing transitive closure of the subproperty and subclass relationships. We presented several algorithms for performing the transitive closure with annotated data, and implemented them in SQL. We performed practical evaluation of the several algorithms over existing data of YAGO and Wordnet knowledge bases for the case of minimum t-norm, and concluded that the matrix and logarithmic versions have better average behaviour than the other versions in the case of annotated data, but the logarithmic method is less reliable. The standard semi-naive evaluation shows poor performance, except when the `subClassOf` instances are in small number. We have shown that our approach for the case of non-annotated data has comparable or better performance than the DLV^{DB} system, and that the overhead imposed by annotations can be significant (from 150% to 350%). The relative behaviour of the compared algorithms carries over from the non-annotated to the annotated versions, except for the recursive algorithm. The recursive annotated transitive closure algorithm is extremely bad behaved, contrary to the non-annotated one.

We plan to extend the experimental evaluation to the large graphs in the Linked Data in order to confirm scalability of the techniques proposed as well as to evaluate the effect of indexing structures. We also would like to evaluate the impact of different t-norms in the running time of our algorithms. Moreover, since our proposal relies on standard database technology we would like to explore vendor specific facilities to improve performance of the developed system as well as increase generality. We also intend to explore alternative memory-based implementations and compare them with the current system.

References

1. Abadi, D.J., Marcus, A., Madden, S., Hollenbach, K.: SW-store: a vertically partitioned dbms for semantic web data management. VLDB 18(2), 385–406 (2009)
2. Biskup, J., Stiefeling, H.: Transitive closure algorithms for very large databases. In: Proceedings of the 14th International Workshop on Graph-Theoretic Concepts in Computer Science, London, UK, pp. 122–147. Springer, Heidelberg (1989)
3. Bonatti, P.A., Hogan, A., Polleres, A., Sauro, L., and: Robust and scalable linked data reasoning incorporating provenance and trust annotations. Journal of Web Semantics (to appear, 2011)
4. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 54–68. Springer, Heidelberg (2002)

5. Damásio, C.V., Medina, J., Ojeda-Aciego, M.: Termination of logic programs with imperfect information: applications and query procedure. J. Applied Logic 5(3), 435–458 (2007)
6. Duan, S., Kementsietsidis, A., Srinivas, K., Udrea, O.: Apples and oranges: a comparison of RDF benchmarks and real RDF datasets. In: Proceedings of the 2011 International Conference on Management of Data, SIGMOD 2011, pp. 145–156. ACM, New York (2011)
7. Guo, Y., Pan, Z., Heflin, J.: Lubm: A benchmark for owl knowledge base systems. J. Web Sem. 3(2-3), 158–182 (2005)
8. Hajek, P.: The Metamathematics of Fuzzy Logic. Kluwer (1998)
9. Hogan, A., Harth, A., Polleres, A.: SAOR: Authoritative Reasoning for the Web. In: Domingue, J., Anutariya, C. (eds.) ASWC 2008. LNCS, vol. 5367, pp. 76–90. Springer, Heidelberg (2008)
10. Hogan, A., Harth, A., Polleres, A.: Scalable authoritative OWL reasoning for the web. Int. J. Semantic Web Inf. Syst. 5(2), 49–90 (2009)
11. Hogan, A., Pan, J.Z., Polleres, A., Decker, S.: SAOR: Template Rule Optimisations for Distributed Reasoning Over 1 Billion Linked Data Triples. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 337–353. Springer, Heidelberg (2010)
12. Ianni, G., Krennwallner, T., Martello, A., Polleres, A.: Dynamic Querying of Mass-Storage RDF Data with Rule-Based Entailment Regimes. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 310–327. Springer, Heidelberg (2009)
13. Ianni, G., Martello, A., Panetta, C., Terracina, G.: Efficiently querying RDF(S) ontologies with answer set programming. J. Log. Comput. 19(4), 671–695 (2009)
14. Ioannidis, Y.E.: On the computation of the transitive closure of relational operators. In: Proceedings of the 12th International Conference on Very Large Data Bases, VLDB 1986, pp. 403–411. Morgan Kaufmann Publishers Inc., San Francisco (1986)
15. Mohri, M.: Semiring frameworks and algorithms for shortest-distance problems. J. Autom. Lang. Comb. 7, 321–350 (2002)
16. Muñoz, S., Pérez, J., Gutierrez, C.: Minimal Deductive Systems for RDF. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 53–67. Springer, Heidelberg (2007)
17. Lopes, N., Polleres, A., Straccia, U., Zimmermann, A.: AnQL: SPARQLing Up Annotated RDFS. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 518–533. Springer, Heidelberg (2010)
18. Nuutila, E.: Efficient transitive closure computation in large digraphs. Acta Polytechnica Scandinavia: Math. Comput. Eng. 74, 1–124 (1995)
19. RDF Semantics. W3C Recommendation, Edited by Patrick Hayes (February 10, 2004),
`http://www.w3.org/TR/2004/REC-rdf-mt-20040210/`
20. Silberschatz, A., Korth, H.F., Sudarshan, S.: Database System Concepts, 6th Edition, 6th edn. McGraw-Hill (2010), Material available at
`http://codex.cs.yale.edu/avi/db-book/`
21. Straccia, U.: A Minimal Deductive System for General Fuzzy RDF. In: Polleres, A., Swift, T. (eds.) RR 2009. LNCS, vol. 5837, pp. 166–181. Springer, Heidelberg (2009)

22. Straccia, U., Lopes, N., Lukacsy, G., Polleres, A.: A general framework for representing and reasoning with annotated semantic web data. In: Fox, M., Poole, D. (eds.) Procs. of AAAI 2010, AAAI Press (2010)
23. Stuckenschmidt, H., Broekstra, J.: Time – Space Trade-Offs in Scaling up RDF Schema Reasoning. In: Dean, M., Guo, Y., Jun, W., Kaschek, R., Krishnaswamy, S., Pan, Z., Sheng, Q.Z. (eds.) WISE 2005 Workshops. LNCS, vol. 3807, pp. 172–181. Springer, Heidelberg (2005)
24. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: A Core of Semantic Knowledge. In: 16th international World Wide Web Conference (WWW 2007), ACM Press, New York (2007)
25. Terracina, G., Leone, N., Lio, V., Panetta, C.: Experimenting with recursive queries in database and logic programming systems. TPLP 8(2), 129–165 (2008)
26. Udrea, O., Recupero, D.R., Subrahmanian, V.S.: Annotated rdf. ACM Trans. Comput. Log. 11(2) (2010)
27. Urbani, J., Kotoulas, S., Oren, E., van Harmelen, F.: Scalable Distributed Reasoning Using MapReduce. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 634–649. Springer, Heidelberg (2009)
28. Zadeh, L.A.: The concept of a linguistic variable and its application to approximate reasoning - parts I, II and III. Inf. Sciences 8(3), 199–249, 301–357, 43–80 (1975)
29. Zimmermann, A., Lopes, N., Polleres, A., Straccia, U.: A general framework for representing, reasoning and querying with annotated semantic web data. In: CoRR, abs/1103.1255 (2011)