Practical RDF Schema reasoning with annotated Semantic Web data

Carlos Viegas Damásio and Filipe Ferreira

October 25, 2011

CENTRIA - FCT/UNL

RDF Meta-Information and Reification

- RDF data is expressed by triples, $<$ Subject, Predicate, Object$>$
- It is useful to add meta-information to RDF data, like:
  Temporal; Confidence; Provenance.
- RDFS defines a way to do this by Reification.
  contact:Name01 rdf:type rdf:Statement .
  contact:Name01 rdf:subject contact:Person .
  contact:Name01 rdf:predicate contact:fullName .
  contact:Name01 rdf:object Eric Miller .

RDF Meta-Information and Reification

- RDF data is expressed by triples, $<$ Subject, Predicate, Object$>$
- It is useful to add meta-information to RDF data, like:
  Temporal; Confidence; Provenance.
- RDFS defines a way to do this by Reification.
  contact:Name01 rdf:type rdf:Statement .
  contact:Name01 rdf:subject contact:Person .
  contact:Name01 rdf:predicate contact:fullName .
  contact:Name01 rdf:object Eric Miller .
- It has no semantic specification for reified data inference.

Annotated RDF(S) data

- An alternative is to extend triples with annotations: <Subject, Predicate, Object>: Annotation
- It has a semantic specification for annotated data inference, based on the ρdf RDFS subset (Straccia et al.)
- ρdf = $\{\texttt{subPropertyOf}, \texttt{subClassOf}, \texttt{type}, \texttt{domain}, \texttt{range}\}$

Annotated RDF(S) data

- An alternative is to extend triples with annotations: <Subject, Predicate, Object>: Annotation
- It has a semantic specification for annotated data inference, based on the ρdf RDFS subset (Straccia et al.)
- ρdf = {`subPropertyOf`,`subClassOf`, `type`, `domain`, `range` }

$$\frac{(A,sp,B),(X,A,Y)}{(X,B,Y)}$$

Annotated RDF(S) data

- An alternative is to extend triples with annotations: <Subject, Predicate, Object>: Annotation
- It has a semantic specification for annotated data inference, based on the ρdf RDFS subset (Straccia et al.)
- ρdf = $\{$`subPropertyOf`,`subClassOf`, `type`, `domain`, `range` $\}$

$$\frac{(A{,}sp{,}B){,}(X{,}A{,}Y)}{(X{,}B{,}Y)}$$

$$\frac{(A{,}sp{,}B){:}v1{,}(X{,}A{,}Y){:}v2}{(X{,}B{,}Y){:}v1 \bigotimes v2}$$

Inference Rules

1. Subproperty

(a) $\frac{(A,sp,B):v1,(B,sp,C):v2}{(A,sp,C):v1 \otimes v2}$

(b) $\frac{(A,sp,B):v1,(X,A,Y):v2}{(X,B,Y):v1 \otimes v2}$

2. Subclass

(a) $\frac{(A,sc,B):v1,(B,sc,C):v2}{(A,sc,C):v1 \otimes v2}$

(b) $\frac{(A,sc,B):v1,(X,type,A):v2}{(X,type,B):v1 \otimes v2}$

3. Typing

(a) $\frac{(A,dom,B):v1,(X,A,Y):v2}{(X,type,B):v1 \otimes v2}$

(b) $\frac{(A,range,B):v1,(X,A,Y):v2}{(Y,type,B):v1 \otimes v2}$

4. Implicit Typing

(a) $\frac{(A,dom,B):v1,(C,sp,A):v2,(X,C,Y):v3}{(X,type,B):v1 \otimes v2 \otimes v3}$

(b) $\frac{(A,range,B):v1,(C,sp,A):v2,(X,C,Y):v3}{(Y,type,B):v1 \otimes v_2 \otimes v3}$

5. Generalization

$$\frac{(X,A,Y):v1,(X,A,Y):v2}{(X,A,Y):v_1 \vee v2}$$

Objectives

- Design and implementation of a database schema to store semantic web data annotated with values of the domain [0,1].
- Implementation using the SQL language with plpgsql support (a procedural language of PostgreSQL) of the classical RDFS inference rules.
- Extension of the the SQL implementation of the inference rules to deal with annotations according to the extended inference rules using $x \bigotimes y = min(x, y)$.
- Testing for correctness and scalability using tailored tests and existing datasets.

Storage Schema

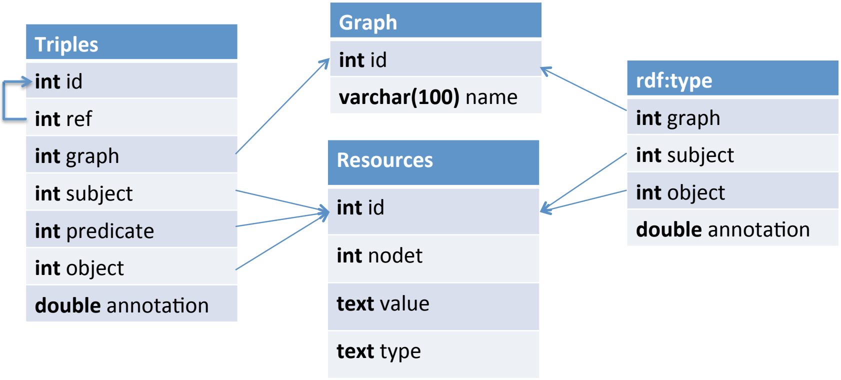


Figura: Annotated RDFS table schema

Fixpoint iteration and rule ordering: ρdf

Rule 2b

$$\frac{(A,sc,B):v1,(X,type,A):v2}{(X,type,B):v1 \otimes v2}$$

Fixpoint iteration and rule ordering: ρdf

Rule 2b
$$\frac{(A,sc,B):v1,(X,type,A):v2}{(X,type,B):v1\otimes v2}$$

Depends on:

Rule 2a
$$\frac{(A,sc,B):v1,(B,sc,C):v2}{(A,sc,C):v1\otimes v2}$$

Fixpoint iteration and rule ordering: ρdf

Rule 2b
$$\frac{(A,sc,B):v1,(X,type,A):v2}{(X,type,B):v1\otimes v2}$$

Depends on:

Rule 2a
$$\frac{(A,sc,B):v1,(B,sc,C):v2}{(A,sc,C):v1\otimes v2}$$

and

Rule 3b
$$\frac{(A,range,B):v1,(X,A,Y):v2}{(Y,type,B)}$$

Fixpoint iteration and rule ordering: ρdf

Rule 2b

$$\frac{(A,sc,B){:}v1,(X,type,A){:}v2}{(X,type,B){:}v1 \otimes v2}$$

Depends on:

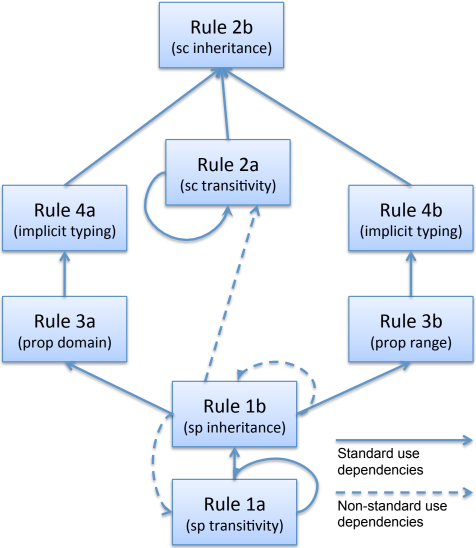
Rule 2a

$$\frac{(A,sc,B){:}v1,(B,sc,C){:}v2}{(A,sc,C){:}v1 \otimes v2}$$

and

Rule 3b

$$\frac{(A,range,B){:}v1,(X,A,Y){:}v2}{(Y,type,B)}$$



Classical non-recursive rule implementation

- Each rule needs only a single query.
- The rule $\frac{(A,sp,B),(X,A,Y)}{(X,B,Y)}$ can be implemented as:

Example

```
INSERT INTO "Triples"(g, s, p, o, a)
(
  SELECT DISTINCT ON (q2.s, q1.o, q2.o) q1.g, q2.s, q1.o, q2.o, 1
  FROM "subPropertyOf" AS q1 INNER JOIN "Triples" AS q2
      ON (q1.s=q2.p)
  WHERE q1.g=i_graph AND q2.g=i_graph
      AND NOT EXISTS (SELECT * FROM "Triples" AS t WHERE
       t.s = q2.s AND t.p=q1.o AND t.o = q2.o AND t.g=i_graph)
);
```

Generalization rule and annotated closure

- If we can derive the same triple with different annotation values, we should derive only the one with the larger annotation value.
- Generalization rule implemented with the MAX aggregate function.

$$\frac{(X,A,Y):v1,(X,A,Y):v2}{(X,A,Y):v1 \vee v2}$$

Generalization rule and annotated closure

- If we can derive the same triple with different annotation values, we should derive only the one with the larger annotation value.
- Generalization rule implemented with the MAX aggregate function.

$$\frac{(X,A,Y):v1,(X,A,Y):v2}{(X,A,Y):v1 \vee v2}$$

- T-norm operation implemented using tnorm function.
- Has input of two double values, returns the minimum value.

Generalization rule and annotated closure

- If we can derive the same triple with different annotation values, we should derive only the one with the larger annotation value.
- Generalization rule implemented with the MAX aggregate function.

$$\frac{(X,A,Y):v1,(X,A,Y):v2}{(X,A,Y):v1 \vee v2}$$

- T-norm operation implemented using tnorm function.
- Has input of two double values, returns the minimum value.

- Triples that already exist in the graph can be infered though other triples with different annotation values.
- We need to guarantee that annotation values for all the existing triples are the maximum possible.
- Solution: Update the annotation value of existing triples.

Annotated non-recursive rule (code skeleton)

Example

```
UPDATE "Triples" as r
SET annotation=d.a
FROM (
   SELECT q1.g, q2.s, q1.o, q2.o, MAX(tnorm(q1.a,q2.a))
   ...
   ) AS d
WHERE (r.s, r.p, r.o, r.g)=(d.s, d.p, d.o, d.g) and r.a<d.a;

INSERT INTO "Triples"(g, s, p, o, a)
(
   SELECT q1.g, q2.s, q1.o, q2.o, MAX(tnorm(q1.a, q2.a)) as annotation
   ...
   GROUP BY q1.g, q2.s, q1.o, q2.o
);
```

Transitive Closure Algorithms

$$r_1 \circ r_2 \quad = \quad \Pi_{r1.sub \texttt{ as } sub,\ r2.obj \texttt{ as } obj}\ \sigma_{r1.obj=r2.sub}(r_1 \times r_2)$$

Naive algorithm

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

Matrix algorithm

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

Transitive Closure Algorithms

$$r_1 \circ r_2 \quad = \quad \Pi_{r1.sub \text{ as } sub,\ r2.obj \text{ as } obj}\ \sigma_{r1.obj=r2.sub}(r_1 \times r_2)$$

Naive algorithm

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

Matrix algorithm

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

- Semi-Naive
- Differential Semi-Naive
- Logarithmic
- PostgreSQL Recursive query

Matrix algorithm implementation

Example

```
LOOP
  INSERT INTO "subClassOf"(
    SELECT q1.g, q1.s, q2.o, q1.a
    FROM "subClassOf"AS q1 INNER JOIN "subClassOf"AS q2 ON
      (q1.o = q2.s)
    WHERE q1.g=i_graph AND q2.g=i_graph
    AND NOT EXISTS (SELECT * FROM "subClassOf"AS sc
      WHERE sc.s = q1.s AND sc.o = q2.o AND sc.g=q1.g)
  );

  GET DIAGNOSTICS nrow = ROW_COUNT;

  IF (nrow=0) THEN
    EXIT;
  END IF;
END LOOP;
```

Matrix annotated algorithm implementation

- Similar to the classical algorithm implementation.
- Uses the MAX and tnorm functions.

Example

```
UPDATE "subClassOf" as r
SET annotation=aux.a
FROM (
  SELECT q1.g, q1.s, q2.o, MAX(tnorm(q1.a,q2.a)) as annotation
  FROM "subClassOf" AS q1 INNER JOIN "subClassOf" AS q2 ON
     (q1.o = q2.s)
  WHERE q1.g=i_graph AND q2.g=i_graph
  GROUP BY q1.g, q1.s, q2.o
) AS aux
WHERE (r.s, r.o, r.g)=(aux.s, aux.o, aux.g) AND r.a<aux.a;
GET DIAGNOSTICS nrow_upd = ROW_COUNT;
```

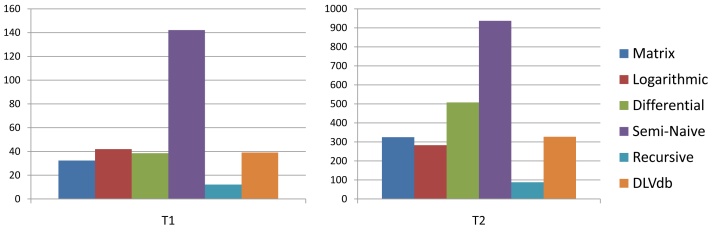
Datasets and Tests

- Tests performed using a Laptop with an Intel i5 2.27GHz processor, 4Gb of RAM and running Windows 7 64-bit.
- Used RDBMS PostgreSQL 9.0.
- Default server configuration.
- Data extracted from the YAGO, YAGO2 and WordNet 2.0 knowledge bases.

	T1	T2	T5	T6
Input Size	0.066M	0.366M	0.417M	1.942M
Output Size	0.599M	3.617M	3.790M	4.947M

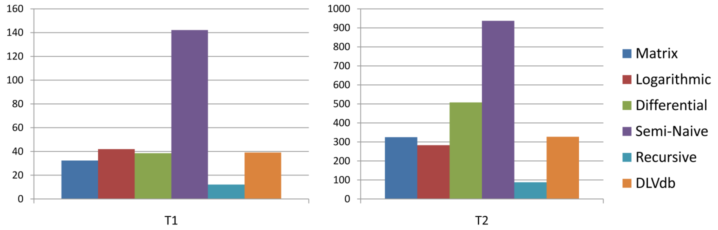
Transitive closure test results

- Results for subclass transitivity tests for classical implementation

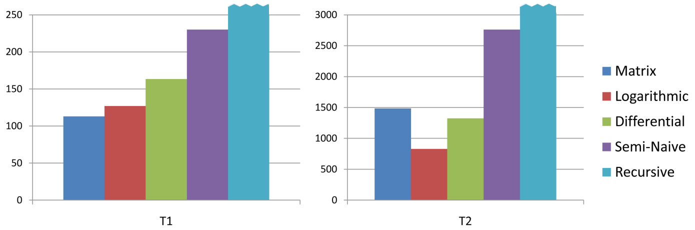


Transitive closure test results

- Results for subclass transitivity tests for classical implementation

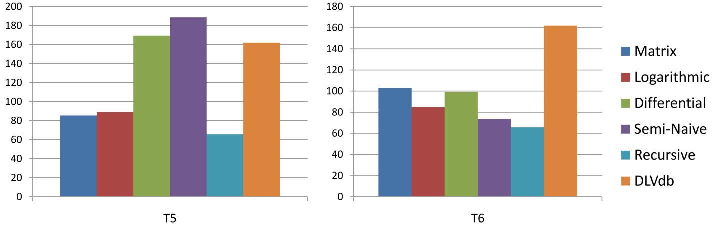


- Results for subclass transitivity tests for annotated implementation



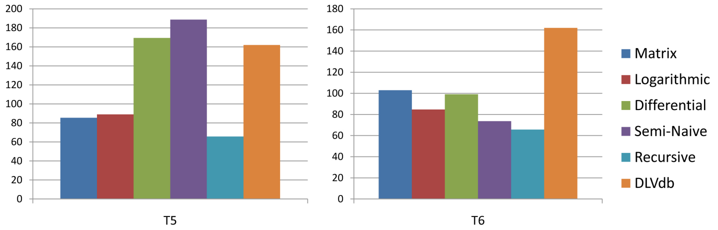
Graph closure test results

- Results for graph closure tests for classical implementation

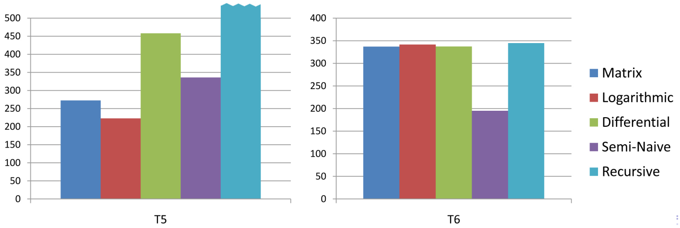


Graph closure test results

- Results for graph closure tests for classical implementation



- Results for graph closure tests for annotated implementation



Conclusions

- We present a full relational database implementation of the annotated RDFS closure rules.
- We propose a rule dependency graph for the ρdf rules, concluding that only recursive rules are the transitive closure rules.

Conclusions

- We present a full relational database implementation of the annotated RDFS closure rules.
- We propose a rule dependency graph for the ρdf rules, concluding that only recursive rules are the transitive closure rules.
- For transitive closure Matrix and Logarithmic methods seem better.
- Annotated reasoning introduces a overhead between 150% and 350%.
- Recent results show that optimization of the database server configuration has significant improvements.

Questions