

Annotated RDF

OCTAVIAN UDREA, DIEGO REFORGIATO RECUPERO, V.S. SUBRAHMANIAN
University of Maryland Institute for Advanced Computer Studies

There are numerous extensions of RDF that support reasoning about uncertainty, reasoning about pedigree, reasoning about time, and so on. In this paper, we present *Annotated RDF* (or **aRDF** for short) in which RDF triples are annotated by members of a partially ordered set (with bottom element) that can be selected in any way desired by the user. We present a formal declarative semantics (model theory) for annotated RDF and develop algorithms to check consistency of **aRDF** theories and to answer queries to **aRDF** theories. We show that annotated RDF captures versions of all the forms of reasoning mentioned above within a single unified framework. We develop a prototype **aRDF** implementation and show that our algorithms work efficiently even on real world data sets containing over 10 million triples.

Categories and Subject Descriptors: F.4.1 [Mathematical logic]: Computational logic; H.2.1 [Logical Design]: Data models; H.2.4 [Systems]: Query processing

General Terms:

Additional Key Words and Phrases: annotated RDF, query processing, view maintenance

1. INTRODUCTION

Since the adoption of “Resource Description Framework” (RDF) as a web recommendation by the W3C, there has been growing interest in using RDF for knowledge representation [Kahan and Koivunen 2001; Carroll et al. 2005; Karvounarakis et al. 2002]. Extensions to RDF have included temporal extensions [Gutiérrez et al. 2005], fuzzy extensions [Dubois et al. 2005; Straccia 2005], provenance management methods [Carroll et al. 2005; Halaschek-Wiener et al. 2006], and others. Extensions along some of these directions have also been made in other semantic web settings [Lukasiewicz and Straccia ; Cali and Lukasiewicz].

An earlier version of this paper [Udrea et al. 2006] appeared in the 2006 European Semantic Web Conference. In the conference version of the paper we presented the syntax and semantics of **aRDF**, algorithms to answer atomic queries (queries of one variable) and evaluate the implementation of our algorithms on a synthetic **aRDF** dataset. We added a number of important features in the current version of the paper. First, we express general conjunctive queries similar to the ones in the RDF query language SPARQL¹ and provide algorithms to efficiently answer such queries. Second, we provide view maintenance algorithms for insert and deletion operations and analyze their correctness and complexity. Third, we perform a thorough experimental evaluation on three different datasets – two real-world **aRDF** theories of 549K and over 12M triples respectively and one synthetically generated dataset ranging from 10K to 10M triples.

Work supported in part by ARO grant DAAD190310202, ARL grants DAAD190320026 and DAAL0197K0135, and NSF grants IIS0329851 and 0205489.

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 2007 ACM 1529-3785/2007/0700-0001 \$5.00

In this paper, we propose an extension of RDF called *Annotated RDF* (or **aRDF** for short) that builds upon *annotated logic* [Kifer and Subrahmanian 1992; Leach and Lu 1996] which has been subsequently used, extended and improved [Fitting 1991] for a wide range of knowledge representation tasks. In **aRDF**, you can start with any partially ordered set that you like as long as it has a bottom element². \mathcal{A} could capture fuzzy or possibilistic values [Carroll et al. 2005; Straccia 2005] or timestamps [Gutiérrez et al. 2005] or - as we shall show - pedigree information or temporal-fuzzy information, and so on.

We present a syntax for **aRDF** in Section 2 - in essence, an **aRDF** triple consists of an ordinary RDF triple together with an annotation (member of \mathcal{A}). We then present a declarative (model-theoretic) semantics for **aRDF**, together with notions of consistency and entailment in Section 3 — unlike ordinary RDF, an **aRDF** theory can be inconsistent and hence we provide a consistency check algorithm, together with a result that whenever the partial order is a lattice, consistency is guaranteed. The syntax and semantics of **aRDF** show that this is a unifying framework that can be used to reason about time, uncertainty, provenance, as well as combinations thereof. Rather than inventing these from scratch, we can use this single framework and instantiate it to specific extensions (old and new) of RDF by the judicious choice of a lattice.

In Section 4, we present the **aRDF** query language, including atomic (one-variable) queries, conjunctive queries, and aggregate queries. We give efficient algorithms to answer these types of queries in Section 5. We also give efficient algorithms for view maintenance in **aRDF** in Section 6. We then present our prototype implementation and experiments in Section 7. Our experiments are rooted in two real-world data sets: the ChefMoz data set (www.chefmoz.org) and the GovTrack data set (www.govtrack.org) containing 549,000 and over 12 million triples respectively. In addition, we conducted experiments with synthetically generated data ranging from 10,000 to 10 million triples. Our experiments show that our framework is very efficient to implement in practice.

2. ARDF SYNTAX

In this section, we define the syntax of **aRDF** triples. We assume the existence of a partially ordered finite set (\mathcal{A}, \preceq) where elements of \mathcal{A} are called *annotations* and \preceq is a partial ordering on \mathcal{A} . We further assume that \mathcal{A} has a unique bottom element. For example, we could have any of the following scenarios:

- (1) \mathcal{A}_{fuzzy} may be the set of all real numbers in the closed interval $[0, 1]$ with the usual “less than or equals” ordering on it.
- (2) $\mathcal{A}_{time} = \mathbf{N}$ could be the set of all non-negative integers (denoting time points) with the usual “less than or equals” ordering on it.
- (3) $\mathcal{A}_{time-int} = \{[x, y] \mid x, y \in \mathbf{N}\}$ could be the set of all time intervals. The interval $[x, y]$ as usual denotes the set of all $t \in \mathbf{N}$ such that $x \leq t \leq y$. The inclusion ordering \subseteq is a partial ordering on this set.

²Suppose (\mathcal{A}, \preceq) is a partially ordered set. $\perp \in \mathcal{A}$ is the “bottom element” of \mathcal{A} iff $\perp \preceq x$ for all $x \in \mathcal{A}$.

- (4) $\mathcal{A}_{pedigree}$ could be an enumerated set consisting of the names of data sources with a partial ordering on them. If $s_1, s_2 \in \mathcal{A}_{pedigree}$, then we could think of $s_1 \preceq s_2$ to mean that s_2 has “better” pedigree than s_1 .
- (5) $\mathcal{A}_{set-pedigree}$ could be the power set of $\mathcal{A}_{pedigree}$ with the Egli-Milner ordering which says that $S_1 \preceq S_2$ iff $(\forall s_1 \in S_1)(\exists s_2 \in S_2)s_1 \sqsubseteq s_2 \wedge (\forall s_2 \in S_2)(\exists s_1 \in S_1)s_1 \sqsubseteq s_2$. Note here that \sqsubseteq is the ordering on $\mathcal{A}_{pedigree}$.
- (6) $\mathcal{A}_{fuztime}$ could be the set of all pairs (x, y) such that $x \in [0, 1]$ is a fuzzy value and y is a time point. The \preceq ordering on $\mathcal{A}_{fuztime}$ can be defined as $(x, y) \preceq (x', y')$ iff $x \leq x'$ and $y \leq y'$.

These are just a few examples of partial orders. All the partial orders above except $\mathcal{A}_{pedigree}$ and $\mathcal{A}_{set-pedigree}$ are complete lattices³. Note that one can construct arbitrary combinations of partial orders by taking the Cartesian Product of two known partial orders and taking the pointwise ordering on the Cartesian Product as shown in the definition of $\mathcal{A}_{fuztime}$. Thus, if we have partially ordered sets $(\mathcal{A}_1, \preceq_1)$ and $(\mathcal{A}_2, \preceq_2)$, then we can generate a new partially ordered set $(\mathcal{A}_1 \times \mathcal{A}_2, \preceq_3)$ where $\mathcal{A}_1 \times \mathcal{A}_2$ is the Cartesian Product of \mathcal{A}_1 and \mathcal{A}_2 and $(x, y) \preceq_3 (x', y')$ iff $x \preceq_1 x'$ and $y \preceq_2 y'$.

Suppose now that (\mathcal{A}, \preceq) is an arbitrary but fixed partially ordered set. As in the case of RDF, we also assume the existence of some arbitrary but fixed set \mathcal{R} of resource names, a set \mathcal{P} of property names, and a set $dom(p)$ of values associated with any property name p .

An *annotated RDF theory* (aRDF-theory for short) is a finite set of triples $(r, p : a, v)$ where r is a *resource* name, p is a *property* name, $a \in \mathcal{A}$ and v is a value (which could also be a resource name).

Note. The reader familiar with RDF Schema will note that this representation also supports RDF Schema triples such as⁴: (i) $(A, rdfs : subclassOf, B)$ which indicates a subclass relationship between classes (which are also resources); (ii) $(X, rdf : type, C)$ which indicates that a resource X is an instance of some class C ; (iii) $(p, rdfs : subPropertyOf, q)$ which denotes a sub-property relation between $p, q \in \mathcal{P}$ ⁵. We denote by $rdfs : subPropertyOf^*$ the transitive closure of $rdfs : subPropertyOf$.⁶

Once \mathcal{R}, \mathcal{P} and $dom(\cdot)$ are fixed, we use the notation $Univ$ to denote the set of all triples (r, p, v) where $s \in \mathcal{R}, p \in \mathcal{P}$ and $v \in dom(p)$. *Throughout the rest of this paper, we will assume that $\mathcal{R}, \mathcal{P}, \mathcal{A}, \preceq, dom(\cdot)$ are all arbitrary, but fixed.*

Definition 2.1. (aRDF theory graph). Suppose \mathcal{O} is an aRDF-theory. An *aRDF theory graph* for \mathcal{O} is a labeled graph (V, E, λ) where

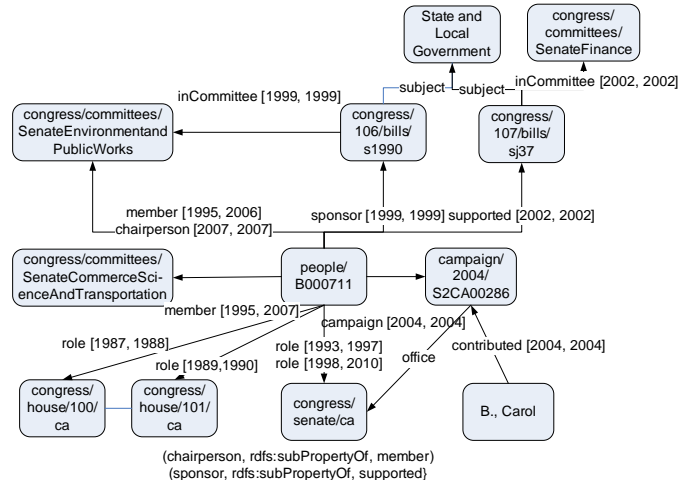
- (1) $V = \mathcal{R} \cup \bigcup_{p \in \mathcal{P}} dom(p)$ is the set of nodes.

³A partially ordered set (X, \preceq) is a complete lattice iff (i) every subset of X has a unique greatest lower bound and (ii) every *directed* subset of X has a unique least upper bound. A set $Y \subseteq X$ is directed iff for all $y_1, y_2 \in Y$, there is an $x \in X$ such that $y_1 \preceq x$ and $y_2 \preceq x$.

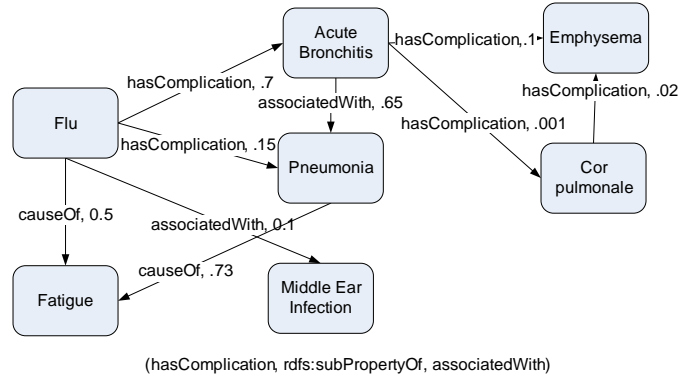
⁴ $rdfs : range$ and $rdfs : domain$ are also possible, as well as any other RDFS construct. However, for the purpose of answering queries, $rdfs : subPropertyOf$ triples are the most important schema triples.

⁵Note we did not require that $\mathcal{P} \cap \mathcal{R} = \emptyset$.

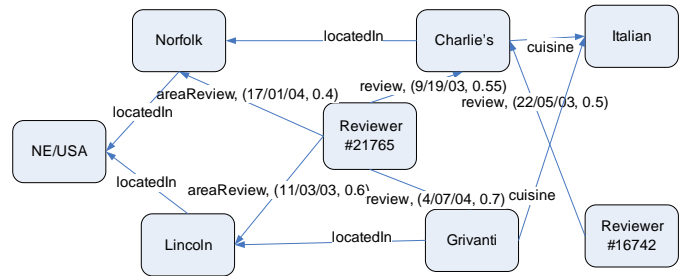
⁶We do not handle RDF’s blank nodes in our framework. However, to date, we have seen very few real world data sets that use blank nodes.



(a) Example aRDF graph annotated with $\mathcal{A}_{time-int}$. Extracted from the GovTrack dataset available at <http://www.govtrack.us>.

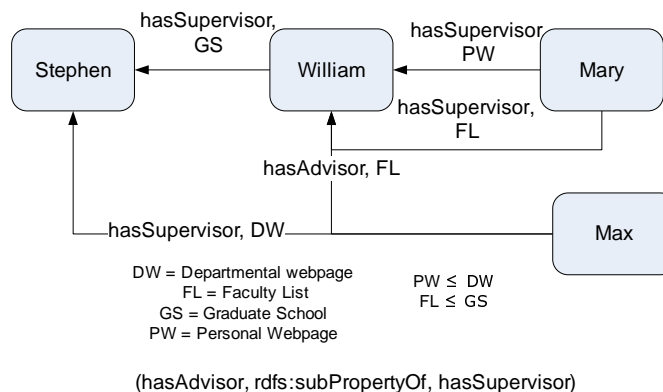


(b) Example aRDF graph annotated with \mathcal{A}_{fuzzy} . aRDF constructed based on information from www.wrongdiagnosis.com.



(c) Example aRDF graph annotated with $\mathcal{A}_{fuztime}$. Extracted from the ChefMoz dataset available at <http://chefmoz.org>.

Fig. 1. Four aRDF theory graphs



(d) Example aRDF graph annotated with $\mathcal{A}_{pedigree}$. Example is purposefully inconsistent to illustrate the aRDF consistency checking algorithm.

Fig. 1 (continued) Four aRDF theory graphs

- (2) $E = \{(r, r') \mid \text{there exists a property } p \text{ such that } (r, p : a, r') \in O\}$ is the set of edges.
- (3) $\lambda(r, r') = \{p : a \mid (r, p : a, r') \in O\}$ is the edge labeling function.

It is easy to see that there is a one-to-one correspondence between aRDF-theories and aRDF-theory graphs. Hence, we will often abuse notation and interchangeably talk about both aRDF theories and aRDF theory graphs.

EXAMPLE 2.2. Figure 1 shows four examples of aRDF theory graphs. Figure 1(a), annotated with elements of $\mathcal{A}_{time-int}$ is extracted from the GovTrack dataset. The dataset consists of approximately 12 million aRDF triples (1.5 GB), containing detailed information about the U.S. Congress and the election campaigns since the early 1980s until the present. The triple $(\text{people/B000711}, \text{role:[1987,1988]}, \text{congress/house/100/ca})$ denotes the fact that the congressperson identified by people/B00711 was a representative of the state of California in the 100th Congress between 1987 and 1988.

Figure 1(b) shows an example aRDF graph constructed manually from information available at www.wrongdiagnosis.com, a website that presents medical information in an ontology-like fashion. The data is annotated with \mathcal{A}_{fuzzy} . The triple $(\text{Flu}, \text{causeOf:0.5}, \text{Fatigue})$ intuitively says that in 50% of cases of Flu, Fatigue is one of the symptoms.

Figure 1(c) shows an example extracted from the ChefMoz dataset, which contains information and reviews of restaurants throughout the world. The dataset consists of approximately 550,000 aRDF triples (220 MB). We used the review information (time and score) from the dataset to annotate the triples. The triple $(\text{Reviewer \#21765}, \text{review: (4/07/04, .7)}, \text{Grivanti})$ denotes the fact that the reviewer with identifier 21765 wrote a review for the Grivanti restaurant on July 4th 2004, giving it a score of .7. In this example, the triples without annotations are assumed to be annotated with the current time and the value 1.

Finally, 1(d) is an example annotated with pedigree information. The example

will be used to illustrate the consistency checking algorithm for *aRDF* in Section 3. In this dataset, there are four sources of information (described in the figure), along with a partial order based on the reliability of the sources. The triple $(Max, hasSupervisor: DW, Stephen)$ denotes the fact that the department webpage (*DW*) lists that Stephen is Max's supervisor.

As in the case of OWL[G. Antoniou 2004], we differentiate between *transitive* and *non-transitive* properties. We assume that all properties in \mathcal{P} are marked transitive or non-transitive. For instance, in Figure 1(d), we consider *hasSupervisor* to be a transitive property⁷. We make this distinction because transitive properties are used very often in the real-world datasets we have analyzed (e.g., *partOf* when discussing congressional organizations, *worksFor* when discussing corporate hierarchy or *causeOf* when discussing diseases and symptoms) and there is no explicit inferential mechanism in RDF for transitivity of general properties (RDF only supports transitivity of *rdfs:subClassOf*).

Definition 2.3 p-Path. Let O be an *aRDF* theory graph, p be a transitive property in O , and suppose $r, r' \in O$ are two nodes. There is a *p-path* between r and r' if there exist triples $t_1 = (r, p_1 : a_1, r_1), \dots, t_i = (r_{i-1}, p_i : a_i, r_i), \dots, t_k = (r_{k-1}, p_k : a_k, r') \in O$ such that for all $i \in [1, k]$ $(p_i, rdfs : subPropertyOf^*, p)$. We will denote a *p-path* Q by the set of triples $\{t_1, \dots, t_k\}$ that form the path; we also say $A_Q = \{a_1, \dots, a_k\}$ is the annotation of the *p-path* Q .

EXAMPLE 2.4. Consider the *aRDF* theory graph shown in Figure 1(d) and suppose the *hasSupervisor* property is transitive. The triples $(Max, hasAdvisor:FL, William)$ and $(William, hasSupervisor:GS, Stephen)$ form a *hasSupervisor-path* (remember that *hasAdvisor* is a subproperty of *hasSupervisor*).

3. ARDF SEMANTICS

In this section, we provide a declarative semantics for *aRDF* theories and study consistency of such theories.

Definition 3.1. An *aRDF-interpretation* I is a mapping from $Univ$ to \mathcal{A} .

The definition of an *aRDF-interpretation* is the natural one that one would expect from annotated logic[Kifer and Subrahmanian 1992]. However, there are two differences that we note here. First, annotated logic in [Kifer and Subrahmanian 1992] assumes that \mathcal{A} is a complete lower semilattice, while we only require that it be a partial order. Second, our definition of satisfaction must take into account, the difference between properties that are transitive and those that aren't. This induces a more complex definition of semantics in our case (given below) than that in [Kifer and Subrahmanian 1992].

Definition 3.2. An *aRDF-interpretation* I satisfies $(r, p : a, v)$ iff $a \preceq I(r, p, v)$. I satisfies an *aRDF-theory* O iff:

(S1) I satisfies every $(r, p : a, v) \in O$.

⁷Although this may not always be the case in the real world, we assume this for the sake of the example.

(S2) For all transitive properties $p \in \mathcal{P}$ and for all p -paths $Q = \{t_1, \dots, t_k\}$ in O , where $t_i = (r_i, p_i : a_i, r_{i+1})$, and for all $a \in \mathcal{A}$ such that $a \preceq a_i$ for all $1 \leq i \leq k$, it is the case that $a \preceq I(r_1, p, r_{k+1})$.

O is *consistent* iff there is at least one aRDF-interpretation that satisfies it. O *entails* $(r, p : a, v)$ iff every aRDF-interpretation that satisfies O also satisfies $(r, p : a, v)$.

The definition of satisfaction and the complex definition of case (S2) above are best illustrated with an example.

EXAMPLE 3.3. *Let O be the aRDF theory graph in Figure 1(b), where $\mathcal{A} = \mathcal{A}_{fuzzy}$. Suppose the associatedWith property is transitive. Let $I_0(t) = (1, now) \forall t \in Univ$. I_0 satisfies O and hence O is consistent. Furthermore, $O \models (Flu, causeOf: .4, Fatigue)$ because for any satisfying interpretation, $0.4 \preceq 0.5 \preceq I(Flu, causeOf, Fatigue)$.*

The intuition behind item (S2) of Definition 3.2 is related to the notion of entailment. For instance, in Figure 1(b) — with associatedWith transitive —, from the triples $(Flu, hasComplication: .7, AcuteBronchitis)$ and $(AcuteBronchitis, associatedWith: .65, Pneumonia)$, we can infer that with a confidence level of at least .65, Flu is associated with Pneumonia since $\forall p \in \mathcal{A}_{fuzzy}$ s.t. $p \preceq .7$ and $p \preceq .65$ (i.e. $\forall p \preceq .65$), $p \preceq I(Flu, associatedWith, Pneumonia)$.

It is immediately clear from Definition 3.2 that aRDF theories can be inconsistent. Consider the aRDF theory graph in Figure 1(d) and assume the *hasSupervisor* property is transitive. We can identify the following sources of inconsistency:

- (1) The triples $(Mary, hasSupervisor:PW, William)$ and $(Mary, hasSupervisor:FL, William)$ ⁸ indicate that for any interpretation I , we cannot have that $PW \preceq I(Mary, hasSupervisor, William)$ and $FL \preceq I(Mary, hasSupervisor, William)$, which contradicts item (S1) from Definition 3.2.
- (2) The presence of the different *hasSupervisor*-paths $\{(Max, hasAdvisor:FL, William), (William, hasSupervisor:GS, Stephen)\}$ and $\{(Max, hasSupervisor:DW, Stephen)\}$ means that for any interpretation I , we cannot have that $FL \preceq I(Max, hasSupervisor, Stephen)$ and $DW \preceq I(Max, hasSupervisor, Stephen)$, thus contradicting item (S2) from Definition 3.2.

We now state a necessary and sufficient condition for checking consistency of an aRDF theory.

THEOREM 3.4. *Let O be an aRDF theory. O is consistent iff:*

- (C1) $\forall p \in \mathcal{P}$ and $\forall r, r' \in \mathcal{R}$ such that there exist distinct $a_1, \dots, a_k \in \mathcal{A}$ and for all $i \in [1, k] \exists (r, p : a_i, r') \in O$, then $\exists a \in \mathcal{A}$ s.t. $\forall i \in [1, k] a_i \preceq a$ AND
- (C2) $\forall p \in \mathcal{P}$ transitive, $\forall r, r' \in \mathcal{R}$, let $\{Q^1, \dots, Q^k\}$ be the set of different p -paths between r and r' and let $\{A_{Q^1}, \dots, A_{Q^k}\}$ be the annotations for these p -paths. Let $B_{Q^i} = \{a \in \mathcal{A} \mid a \preceq a' \forall a' \in A_{Q^i}\}$. Then $\exists a \in \mathcal{A}$ s.t. $\forall b \in \bigcup_{i \in [1, k]} B_{Q^i}, b \preceq a$ ⁹.

⁸The presence of such triples is reasonable since it indicates the same information was obtained from different sources for which we cannot compare the pedigree according to the partial order given.

⁹Note that (C2) implies (C1) when p is transitive, since paths of length 1 are possible.

Proof. Let O be an **aRDF** theory that meets conditions (C1) and (C2) above. Then we can build a satisfying interpretation as follows:

- For any set of triples that match (C1), assign $I(r, p, r') = a$. For any other triple $(r, p : a, v) \in O$, assign $I(r, p, v) = a$.
- For any transitive property p and pair of resources r, r' that match (C2), assign $I(r, p, r') = a$.

It is straightforward to show that the above interpretation satisfies Definition 3.2.

Conversely, let O be a consistent theory. Let I be a satisfying interpretation. We need to show that conditions (C1) and (C2) hold.

To see why condition (C1) holds, suppose $p, r, r', a_1, \dots, a_k$ are as in (C1) and suppose $(r, p : a_i, r') \in O$. Then by condition (S1) in the definition of satisfaction, $a_i \preceq I(r, p, v)$ for all $1 \leq i \leq k$. In this case, we can take a to be $I(r, p, v)$.

To see why condition (C2) holds, suppose p is transitive, $\{Q^1, \dots, Q^k\}$ is the set of different p -paths between resources r and r' , and $\{A_{Q^1}, \dots, A_{Q^k}\}$ are the annotations for these p -paths. Then, by condition (S2) in the definition of satisfaction, for each annotation a_j^i in path Q^i , $a_j^i \preceq I(r, p, v)$. Therefore, $I(r, p, v)$ is an upper bound for the sets B_{Q^i} and hence can serve as the annotation a in (C2).

The following result states that if we require \mathcal{A} to be a partial order with a top element¹⁰, then we are guaranteed consistency.

COROLLARY 3.5. *Let \mathcal{A} be a partial order with a top element. Then any **aRDF** theory O annotated w.r.t. \mathcal{A} is consistent.*

The justification is immediate, since the interpretation that maps every triple in $Univ$ to the top element satisfies any **aRDF** theory.

Theorem 3.4 provides an immediate algorithm for checking the consistency of **aRDF** theories. We present this algorithm in Figure 2.

EXAMPLE 3.6. *Let O the **aRDF** theory graph in Figure 1(d). When we run our consistency check algorithm and execution reaches line 4 with $(r, p, r') = (Mary, hasSupervisor, William)$, $A = \{PW, FL\}$ from line 2. Since $\nexists a \in \mathcal{A}$ s.t. $PW, FL \preceq a$, the algorithm will determine that the theory is inconsistent.*

*Now consider the same **aRDF** theory without the triple $(Mary, hasSupervisor:PW, William)$. In this case, the algorithm will proceed to the loop starting on line 6. However, for the iteration for which $p = hasSupervisor$ on line 6 and $(r, r') = (Max, Stephen)$ on line 9, the set P' will contain the two possible $hasSupervisor$ -paths from Max to $Stephen$ detailed in Example 3.3. Then on line 12, $A = \{\{DW\}, \{FL, GS\}\}$ and on line 13 $B = \{DW, FL\}$ and since $\nexists a \in \mathcal{A}$ s.t. $DW, FL \preceq a$, the algorithm will return *False* on line 14.*

The following result states the correctness of our consistency check algorithm.

PROPOSITION 3.7 CONSISTENCY CHECK CORRECTNESS. *The **aRDF** consistency on input $(O, \mathcal{A}, \preceq)$ returns *True* iff O is consistent.*

Proof. The loop on lines 1–5 corresponds to condition (C1) in Theorem 3.4; lines 6–17 correspond to condition (C2) of the same theorem. The algorithm uses

¹⁰An element $\top \in \mathcal{A}$ is a “top” element if $x \preceq \top$ for all $x \in \mathcal{A}$.

Algorithm aRDFconsistency(O, \mathcal{A}, \preceq)

Input: aRDF theory O and annotation (\mathcal{A}, \preceq) .

Output: *True* if O is consistent, *False* otherwise.

Notation: For a property p we write $SP(p) = \{q \in \mathcal{P} \mid (q, rdfs : subPropertyOf^*, p)\}$. We denote by $O|_p$ the restriction of the aRDF graph O to triples labeled with properties in $SP(p)$. $N(O)$ denotes the set of nodes in the aRDF theory graph O .

```

1. for  $(r, p, r') \in \{(r, p, r') \mid \exists a \in \mathcal{A} \text{ s.t. } (r, p : a, r') \in O\}$  do
2.    $A \leftarrow \{a \in \mathcal{A} \mid (r, p : a, r') \in O\}$ ;
3.   if  $|A| > 1$  then
4.     if  $\nexists a \in \mathcal{A} \text{ s.t. } \forall a' \in A, a' \preceq a$  return False;
5.   end
6.   for  $p \in \mathcal{P}$  transitive do
7.      $O' \leftarrow O|_p$ ;
8.      $P \leftarrow \{\text{paths } Q \subseteq O' \mid \nexists Q' \subseteq O' \wedge Q' \supset Q\}$ ;
9.     for  $(r, r') \in N(O') \times N(O')$  do
10.       $P' \leftarrow \{Q \in P \mid r, r' \text{ are the first and last node respectively in } Q\}$ ;
11.      if  $|P'| > 0$  then
12.         $A \leftarrow \{A_Q \mid Q \in P'\}$ ;
13.         $B \leftarrow \{b \in \mathcal{A} \mid \exists A_Q \in A \text{ s.t. } \forall a \in A_Q, b \preceq a\}$ ;
14.        if  $\nexists a \in \mathcal{A} \text{ s.t. } \forall b \in B, b \preceq a$  then return False;
15.      end
16.    end
17.  end
18. return True;

```

Fig. 2. Consistency checking algorithm for aRDF theories

the fact that if property (C2) in Theorem 3.4 holds for maximal p -paths, then it will also hold for shorter p -paths. This result follows directly from Definition 3.2 and the definition of a partial order.

The consistency check algorithm runs in polynomial time as shown below.

PROPOSITION 3.8 CONSISTENCY CHECK COMPLEXITY. *Let O be an aRDF theory graph and let $n = |N(O)|$, let $e = |O|$ and let $p = |\mathcal{P}|$. Let (\mathcal{A}, \preceq) be a partial order and let $a = |\mathcal{A}|$ ¹¹. Then aRDFconsistency(O, \mathcal{A}, \preceq) is $\mathcal{O}(p \cdot (n^3 \cdot e + n \cdot a^2))$.*

The result follows from the loop on lines 6—17. For any transitive property, we first compute the set of all maximal paths in $O|_p$ (line 8). Since we have to keep the paths in memory (and not only their cost), this operation can be performed in at most $n^3 \cdot e$ steps in a modified version of Floyd’s algorithm [Floyd 1962] that records the paths explored. The loop on line 9 iterates through all the maximal paths found — there can be at most $2n$ of them. For each such path, we compute the set A (line 12), which takes at most e steps, since any maximal path is of length less than or equal to e . The size of each set A is bounded by a and the number of maximal paths for the entire graph is at most $\mathcal{O}(n)$, meaning line 13 will be run at most $\mathcal{O}(n \cdot a^2)$ times. Line 14 is run at most $\mathcal{O}(n \cdot a^2)$ times as well, since $|B|$ is bounded by a .

¹¹We assume without loss of generality that $a < e$, since we can use at most one annotation for each edge.

4. ARDF QUERY LANGUAGE

In this section, we define the aRDF Query Language. We start by discussing *simple aRDF queries* – annotated triples in which any of the subject, property, value or annotation can be either constant or variable. We provide algorithms for the cases in which only one element of the annotated triple is a variable, and then show how these methods can be extended to multiple variables. Finally, we discuss general conjunctive queries.

4.1 Simple queries

We assume the existence of sets of variables ranging over resources, properties, values and \mathcal{A} . A term over one of these sets is either a member of that set or a variable ranging over that set. An *aRDF query* is a triple $(R, P : A, V)$ where R, P, A, V are all terms over resources, properties, annotations and values respectively. An aRDF query of the above form is atomic if at most one term in it is a variable.

EXAMPLE 4.1. Consider the aRDF theory graphs in Figures 1(a)–(c). The following are atomic aRDF queries:

- What committees was *people/B000711* a member of between 1997 and 2001? This is expressed as: $(\text{people}/B000711, \text{member}: [1997, 2001], ?v)$.
- What conditions is *Flu* associatedWith in at least 10% of cases (assuming has-Complication and associatedWith are transitive)? This can be expressed as: $(\text{Flu}, \text{associatedWith}: .1, ?v)$.
- What reviewers gave the restaurant *Grivanti* scores of .5 or higher after 01/01/2004? This can be expressed as: $(?s, \text{review}: (01/01/2004, .5), \text{Grivanti})$.

Definition 4.2 *Semi-unifiable aRDF queries.* Suppose θ is a substitution. Two simple aRDF queries $(r, p : a, v)$, $(r', p' : a', v')$ are θ semi-unifiable iff $r\theta = r'\theta \wedge p\theta = p'\theta \wedge v\theta = v'\theta$.

As usual, $r\theta$ denotes the application of θ to r . Note that the definition of semi-unifiable aRDF queries also applies to aRDF triples as they are also simple aRDF queries.

Definition 4.3 *Query answer.* Let O be a consistent aRDF theory and let $q = (r_q, p_q : a_q, v_q)$ be a simple aRDF query on O . Let $A_O(q) = \{(r, p : a, v) \mid (r, p : q, v) \text{ contains no variables and } (r_q, p_q : a_q, v_q) \text{ is semi-unifiable with } q \text{ and } O \models (r, p : a, v) \wedge ((a_q \text{ is a variable}) \vee (a_q \preceq a))\}$. The *answer* to q is defined as $Ans_O(q) = \{(r, p : a, v) \in A_O(q) \mid \exists S \subseteq Ans_O(q) - \{(r, p : a, v)\} \text{ s.t. } S \models (r, p : a, v)\}$.

$A_O(q)$ consists of all ground (i.e. variable-free) instances of q that are entailed by O . However, $A_O(q)$ may contain redundant triples – for example, using our *time – int* partial ordering, if $(r, p : [1, 100], v)$ is in $A_O(q)$, then there is no point including redundant triples such as $(r, p : [1, 10], v)$ in it. $Ans_O(q)$ eliminates all such redundant triples from $A_O(q)$. Note that there is a one-to-one correspondence between elements of $A_O(q)$ and a set $\Theta_O(q)$ of substitutions, such that $\forall \theta \in \Theta_O(q), \exists!(r, p : a, v) \in A_O(q)$ such that $(r, p : a, v) = (r_q, p_q : a_q, v_q)\theta$.

EXAMPLE 4.4. Consider the queries in Example 4.1. The answers are:

- $Ans_O(q) = \{(people/B000711, member: [1995, 2006], congress/committees/Senate-EnvironmentAndPublicWorks), (people/B000711, member: [1995, 2007], congress/committees/SenateCommerceScienceAndTransportation)\}$. Note that the answer does not include for instance $(people/B000711, member: [1997, 2001], congress/committees/Senate-EnvironmentAndPublicWorks)$ since it is already entailed by a triple in the answer.
- $Ans_O(q) = \{(Flu, associatedWith: .65, Pneumonia), (Flu, hasComplication: .1, Emphysema), (Flu, hasComplication: .7, AcuteBronchitis)\}$.
- $Ans_O(q) = \{(Reviewer \#21765, review: (4/07/04, .7), Grivanti)\}$.

The following result specifies a condition that must hold when O entails a ground aRDF triple.

THEOREM 4.5. *Let O be a consistent aRDF theory and let $(r, p : a, v)$ be an aRDF triple. $O \models (r, p : a, v)$ iff one of the following conditions holds:*

- (E1) $\exists (r, p : a_1, v), \dots, (r, p : a_k, v) \in O$ and let A be the set of values a' such that $a_i \preceq a' \forall i \in [1, k]$ ($|A| \geq 1$ since O is consistent). Then $\forall a' \in A, a \preceq a'$.
- (E2) $\exists p$ -paths Q^1, \dots, Q^k between r and v . Let $B_{Q^i} = \{b \in \mathcal{A} | b \preceq a' \forall a' \in A_{Q^i}\}$. Let A be the set of values a' such that $\forall b \in \bigcup_{i \in [1, k]} B_{Q^i}, b \preceq a'$ ($|A| \geq 1$ since O is consistent). Then $\forall a' \in A, a \preceq a'$.

Proof. Assume that none of (E1, E2) holds. Then we are in one of the following cases:

- There is no edge labeled with p or a subproperty of p or a p -path between r, v . Then for any satisfying interpretation that has $I(r, p, v) \neq \perp$, the interpretation I' s.t. $I'(t) = I(t) \forall t \in Univ - \{(r, p, v)\}$ and $I'(r, p, v) = \perp$ would also be a satisfying interpretation that implies $O \not\models (r, p : a, v)$.
- $\exists (r, p : a_1, v), \dots, (r, p : a_k, v) \in O$ and $\exists a' \in A, a' \preceq a$. Then for any satisfying interpretation I , we can construct I' that differs from I in that $I'(r, p, v) = a'$; I' is also a satisfying interpretation, that does not satisfy $(r, p : a, v)$, which implies $O \not\models (r, p : a, v)$.
- The case where there exist p -paths is similar to the case in which there exist edges between r and v .

Given an theory O , we can infer new triples from O using the following two operators, f_1, f_2 :

- (1) $f_1(O) = \{(r, p : a, v) | \exists (r, p : a_1, v), (r, p' : a_2, v) \in O \text{ s.t. } (p', rdfs : subPropertyOf^*, p) \wedge a \text{ is a minimal upper bound}^{12} \text{ of } a_1, a_2\}$.
- (2) $f_2(O) = \{(r, p : a, v) | \exists (r, p' : a_1, r'), (r', p'' : a_2, v) \in O \text{ s.t. } (p', rdfs : subPropertyOf^*, p) \wedge (p'', rdfs : subPropertyOf^*, p) \wedge (\forall a' \in \mathcal{A}, (a' \preceq a_1 \wedge a' \preceq a_2) \Rightarrow (a' \preceq a)) \wedge (a \text{ minimal with these properties w.r.t. } \preceq)\}$.

Let $\mu(O) = f_1(O) \cup f_2(O)$.

PROPOSITION 4.6 CLOSURE OF O . *μ is a monotonic operator, i.e. if $O_1 \subseteq O_2$ then $\mu(O_1) \subseteq \mu(O_2)$. Hence, by the Tarski-Knaster theorem, it has a least fixpoint denoted by $lfp(O)$ called the closure of O .*

¹² a is an minimal upper bound of a_1, a_2 iff $a_1 \preceq a$ and $a_2 \preceq a$ and there is no other a' such that $a' \preceq a$ and $a_1, a_2 \preceq a'$.

EXAMPLE 4.7. Let O be the **aRDF** theory in Figure 1(b). Besides the triples in O , $\text{lfp}(O)$ also contains $(\text{Flu}, \text{associatedWith} : .65, \text{Pneumonia})$, $(\text{Flu}, \text{hasComplication} : .1, \text{Emphysema})$, $(\text{Flu}, \text{hasComplication} : .001, \text{CorPulmonale})$.

The following result is a necessary and sufficient condition for entailment by an **aRDF** theory.

PROPOSITION 4.8. Let O be an **aRDF** theory. $O \models (r, p : a, v)$ iff $(r, p : a, v) \in \text{lfp}(O)$ or $\exists(r', p' : a', v') \in \text{lfp}(O)$ s.t. $\{(r', p' : a', v')\} \models (r, p : a, v)$.

Proof. Follows from Definition 4.6, which corresponds to the conditions in Theorem 4.5. Here, $\mu(O)$ has been defined to be the set of triples that can be inferred by conditions (E1),(E2) of Theorem 4.5 in exactly one step (i.e. considering only pairs of triples). $\mu(O)$ is then augmented at every step until any triple entailed by O is either contained in $\text{lfp}(O)$ or trivially entailed by a triple in the fixpoint.

The above proposition provides an immediate algorithm to answer ground atomic queries to an **aRDF** theory. The next proposition will provide us with a mechanism to answer atomic queries.

PROPOSITION 4.9. Let O be a consistent **aRDF** theory and q a query on O . Then the following hold:

- (1) (Soundness) $\text{Ans}_q(O) \subseteq \text{lfp}(O)$.
- (2) (Completeness) For all substitutions θ such that $q\theta$ is ground and $O \models q\theta$, $\text{Ans}_q(O) \models q\theta$.

Proof. From Proposition 4.8, we know that all possible answers to a query must either be in $\text{lfp}(O)$ or entailed by a triple in $\text{lfp}(O)$. Therefore, $\forall(r, p : a, v) \in \text{Ans}_q(O) - \text{lfp}(O)$, exists $(r, p' : a', v) \in \text{lfp}(O)$ s.t. $\{(r, p' : a', v)\} \models (r, p : a, v)$. On one hand, $(r, p' : a', v) \notin \text{Ans}_q(O)$ since $\text{Ans}_q(O)$ is maximal w.r.t. entailment. On the other hand, by Definition 4.3, $(r, p : a, v) \in \text{Ans}_q(O) \Rightarrow (r, p' : a', v) \in \text{Ans}_q(O)$ should also be in the answer. We have a contradiction, therefore, $\text{Ans}_q(O) \subseteq \text{lfp}(O)$. Intuitively, the closure $\text{lfp}(O)$ is the minimal set of triples entailed by O w.r.t. entailment within the set $\mu(O)$.

The completeness of $\text{Ans}_q(O)$ results directly from the definition of a query answer. Since $O \models q\theta$, then the ground triple $q\theta$ must be in $A_q(O)$ (Definition 4.3). From the definition of $\text{Ans}_q(O)$, $q\theta$ is either in $\text{Ans}_q(O)$ or entailed by it.

The above proposition gives us a very simple algorithm for answering simple queries (which we will call *naiveSimpleAnswer*). Recall that simple queries are atomic queries with at most one variable in it.

Algorithm *naiveSimpleAnswer*.

- (1) Consider query $q = (r, p : a, v)$ on **aRDF** theory O . Compute $\text{lfp}(O)$.
- (2) $A \leftarrow \{(r', p' : a', v') \in \text{lfp}(O) \mid (r', p' : a', v') \text{ semi-unifiable with } q \wedge ((a \text{ is a variable}) \vee (a \preceq a'))\}$.
- (3) Eliminate from A triples $(r, p : a, v)$ entailed by subsets of $A - \{(r, p : a, v)\}$.

However, we can do much better by avoiding the costly computation of $\text{lfp}(O)$, as we will show in Section 5.

4.2 Conjunctive queries

Conjunctive queries are simply sets of simple aRDF queries that have common variables.

Definition 4.10 Conjunctive query. A conjunctive query Q is a set of simple queries such that for any simple query $q \in Q$, there exists a variable v in q that also appears in another simple query $q' \in Q, q' \neq q$.

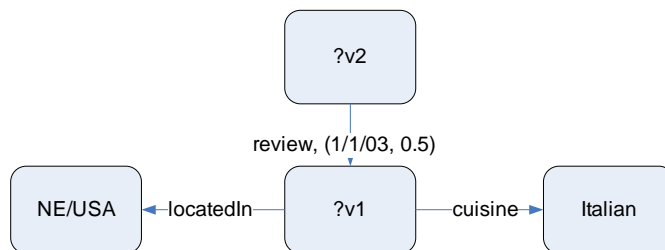


Fig. 3. Example aRDF conjunctive query graph

Note that a conjunctive query is basically a partially instantiated aRDF theory graph. The condition imposed on the query is that the corresponding query graph is *connected*. This requirement is best illustrated with an example.

EXAMPLE 4.11. Consider the aRDF theory in Figure 1(c). The following is a conjunctive query over this theory: which Italian restaurants located in the NE USA had a review with an annotation value a such that $(1/1/2003, .5) \preceq a$? Since we are using less-than for the partial order of both dates and review scores, this means we are looking for Italian restaurants in the NE USA with reviews that appeared on or after January 1 2003 and where the review score is higher than .5. The query can be expressed as $Q = \{(?v1, cuisine, Italian), (?v1, locatedIn, NE/USA), (?v2, review: (1/1/03, .5), ?v1)\}$. A graphical representation of the query is given in Figure 3. We remind the reader that triples not annotated in this example are assumed to be annotated with $(now, 1)$ and that *locatedIn* is a transitive property. Note that the natural language version of the query also asks for a projection operation – we only want the restaurants, hence the possible substitutions for $?v1$. Since projection can be easily performed once the set of possible substitutions for $?v1$ and $?v2$ is computed, we do not explicitly define this operation in the paper.

Observe that this conjunctive query does meet the restriction in Definition 4.10, since $?v1$ appears in all the simple queries. However, the query $\{(?v1, cuisine, Italian), (?v2, cuisine, Italian)\}$ is not a valid conjunctive query since the two simple queries are not linked by any variable.

We also point out that the aRDF conjunctive queries are very similar to SPARQL query patterns, albeit without the syntactic sugar. The main difference between aRDF queries and SPARQL queries is the lack of support for annotations in SPARQL. If we ignore annotations, the query Q can be expressed as the following SPARQL graph pattern:

SELECT FROM G ?v1, ?v2 WHERE

{?v1 cuisine Italian . ?v1 locatedIn NE/USA . ?v2 review ?v1}

If we were to represent the same information in the aRDF theory in Figure 1(c) in standard RDF, we could do so by using reification. For instance, the aRDF statement (Reviewer #21765, review: (4/07/04, .7), Grivanti) can be represented as the set of RDF statements:

(_: a, rdf : subject, Reviewer#21765),
 (_: a, rdf : object, Grivanti),
 (_: a, rdf : predicate, review),
 (_: a, date, 4/07/04),
 (_: a, score, .7)

In this representation, $_: a$ represents a blank RDF node (a node without label), which we have temporarily assigned the label a to distinguish it from other blank nodes in writing. In this case, the SPARQL query pattern corresponding to Q is:

SELECT FROM G ?v1, ?v2 WHERE

{?v1 cuisine Italian . ?v1 locatedIn NE/USA .

_: x rdf : subject ?v2 . _: x rdf : object ?v1 .

_: x rdf : predicate review . _: x date ?date . _: x score ?score .

FILTER(xsd : dateTime(?date) >= xsd : dateTime("4/07/04")) .

FILTER(xsd : float(?score) >= 0.7)}

Definition 4.12 *Conjunctive query answer.* Let $Q = \{q_1, \dots, q_n\}$ be a conjunctive aRDF query. Let $\Theta_O(Q) = \{(\theta_1, \dots, \theta_n) \in \Theta_O(q_1) \times \dots \times \Theta_O(q_n) \mid \theta_1 \cup \dots \cup \theta_n \text{ is consistent}^{13}\}$. We define $A_O(Q) = \{\{e_1, \dots, e_n\} \mid \exists (\theta_1, \dots, \theta_n) \in \Theta_O(Q) \text{ s.t. } \forall i \in [1, n], e_i = q_1 \theta_i\}$. The answer to Q is $Ans_O(Q) = \{\{e_1, \dots, e_n\} \in A_O(Q) \mid (A_O(Q) - \{\{e_1, \dots, e_n\}\}) \not\models \{e_1, \dots, e_n\}\}$.

For a conjunctive query Q , one element of the answer is a set of aRDF triples and each element of this latter set is an answer to one of the simple queries in Q . $Ans_O(Q)$ has the same purpose as for simple queries, namely to eliminate redundant elements from the answer.

EXAMPLE 4.13. Consider the query Q in Example 4.11. The aRDF theory in Figure 1(c) contains two answers to this query (remember that *locatedIn* is a transitive property):

— $\{(Grivanti, locatedIn, NE/USA), (Grivanti, cuisine, Italian), (Reviewer \#21765, review: (4/07/04, .7), Grivanti)\}$

— $\{(Charlie's, locatedIn, NE/USA), (Charlie's, cuisine, Italian), (Reviewer \#16742, review: (22/05/03, .5), Charlie's)\}$

Definition 4.12 also provides a naive method of answering a conjunctive query Q (which we will call *naiveConjunctiveAnswer*).

Algorithm *naiveConjunctiveAnswer.*

¹³We assume, as is frequently done in the unification literature [Martelli and Montanari 1982], that a substitution can be viewed as a system of equations and that a set of substitutions is compatible iff the union of the set of equations corresponding to each substitution is a solvable system of equations.

- (1) Compute the substitutions (answers) $\Theta_O(q)$ for each of the simple queries $q \in Q$.
- (2) Compute the cartesian product $\Theta' = \prod_{q \in Q} \Theta_Q(q)$.
- (3) Select only those elements in $(\theta_1, \dots, \theta_n)$ for which $\theta_1 \cup \dots \cup \theta_n$ is consistent.
- (4) Compute the set of answers $A_O(Q)$ by applying each remaining substitution in Θ' to Q .
- (5) Eliminate the redundant answers from $A_O(Q)$ to obtain $Ans_O(Q)$.

However, computing the cartesian product in step (2) is potentially very time consuming, especially for conjunctive queries with many variables. We will give two more efficient methods in Section 5.

5. ARDF QUERY PROCESSING ALGORITHMS

In the previous section, we defined the types of queries supported by aRDF. In this section, we provide efficient algorithms to answer these types of queries. We start by looking at the problem of atomic queries and then show how the atomic query answer algorithms can be generalized to simple queries with more than one variable. We then give two distinct methods of answering conjunctive queries and show how to process these results to answer aggregate queries.

5.1 Answering atomic queries

Although the closure of an aRDF theory gives a simple method of computing the answer to queries, the computation of $\text{lfp}(O)$ is potentially very expensive. In fact, we show that we can do much better by building only those parts of the closure that are of interest to the given query. We start by focusing on atomic queries – i.e., simple queries with only one variable. The algorithm for queries of type $q = (r, p : a, ?v)$ is given in Figure 4; computing the answers to atomic queries of type $q = (?r, p : a, v)$ is almost identical (with the proper notation change) and therefore omitted.

EXAMPLE 5.1. *Consider the aRDF theory graph in Figure 1(b) and the query (Flu, associatedWith: 0.1, ?v). Since associatedWith is transitive, the algorithm will go on the second branch, starting at line 10. The loop on line 11 iterates through all the values reachable through associatedWith-paths from Flu, which are exactly {AcuteBronchitis, Pneumonia, Emphysema, CorPulmonale}. Let us consider the second iteration, where $v' = \text{Pneumonia}$. There are two associatedWith paths between Flu and Pneumonia. For the first path, going through AcuteBronchitis, $A(Q^1) = \{.7, .65\}$. For the second, direct path, $A_{Q^2} = \{.15\}$. Therefore B on line 12 is exactly the interval $(0, .65]$ and $C = [.65, 1]$. As a result, on line 14, $D = \{.65\}$ and the triple (Flu, associatedWith: .65, Pneumonia) is added to the answer.*

The following theorem states that *atomicAnswerV* is correct and runs in polynomial time.

PROPOSITION 5.2. *Let O be an aRDF theory graph and let n be the number of vertices in the theory graph O , let $e = |O|$ and let $p = |\mathcal{P}|$. Let (\mathcal{A}, \preceq) be a partial order and let $a = |\mathcal{A}|$. Then the following hold:*

- (1) *atomicAnswerV($O, \mathcal{A}, \preceq, q$) returns $Ans_O(q)$.*

Algorithm atomicAnswerV($O, \mathcal{A}, \preceq, q$)**Input:** Consistent aRDF theory O , annotation (\mathcal{A}, \preceq) and query $q = (r_q, p_q : a_q, ?v)$.**Output:** $Ans_O(q)$.**Notation:** For a property p we write $SP(p) = \{q \in \mathcal{P} \mid (q, rdfs : subPropertyOf^*, p)\}$. We denote by $O|_p$ the restriction of the aRDF graph O to triples labeled with properties in $SP(p)$.

1. $O \leftarrow O|_{p_q}$;
2. $Ans \leftarrow \emptyset$;
3. **if** p_q is non-transitive **then**
4. **for** $(r_q, p', v') \in \{(r_q, p' : a', v') \in O\}$ **do**
5. $A \leftarrow \{a' \in \mathcal{A} \mid (r_q, p' : a', v') \in O\}$;
6. $B \leftarrow \{b \in \mathcal{A} \mid \forall a \in A, a \preceq b\}$;
7. $C \leftarrow \{c \in B \mid \nexists c' \in B, c' \neq c \text{ s.t. } c' \preceq c\}$;
8. $Ans \leftarrow Ans \cup \{(r_q, p' : c, v') \mid c \in C \wedge a_q \preceq c\}$;
9. **end**
10. **else if** p_q transitive **then**
11. **for all** v' s.t. $\exists Q^1, \dots, Q^k$ p -paths from r_q to v' **do**
12. $B \leftarrow \{b \in \mathcal{A} \mid \exists i \in [1, k] \text{ s.t. } \forall a' \in A_{Q^i}, b \preceq a'\}$;
13. $C \leftarrow \{c \in \mathcal{A} \mid \forall b \in B, b \preceq c\}$;
14. $D \leftarrow \{d \in C \mid \nexists d' \in C, d' \neq d \text{ s.t. } d' \preceq d\}$;
15. $Ans \leftarrow Ans \cup \{(r_q, p_q : d, v') \mid d \in D \wedge a_q \preceq d\}$;
16. **end**
17. **end**
18. **return** Ans ;

Fig. 4. Answering atomic aRDF queries $(r_q, p_q : a_q, ?v)$

(2) $atomicAnswerV(O, \mathcal{A}, \preceq, q)$ is $\mathcal{O}(n^2 \cdot e + n \cdot e \cdot a^2)$.

Proof. Algorithm correctness follows directly from Theorem 4.5. Lines 3—9 correspond to condition (E1), whereas lines 10—17 correspond to condition (E2).

The complexity result is given by the loop on lines 10—17. We start by determining all values reachable by p_q -paths from r_q and the corresponding p_q paths. This process takes at most $\mathcal{O}(n^2 \cdot e)$ steps. Since there are at most $\mathcal{O}(n)$ p_q -paths originating from r_q , each with at most $\mathcal{O}(e)$ edges and the annotation for each path is bounded by a , line 12 will be run at most $\mathcal{O}(n \cdot e \cdot a^2)$ times. Since the sizes of B, C, D are all bounded by a , the same result holds for lines 13—15.

Algorithm atomicAnswerP($O, \mathcal{A}, \preceq, q$)**Input:** Consistent aRDF theory O , annotation (\mathcal{A}, \preceq) and query $q = (r_q, ?p : a_q, v_q)$.**Output:** $Ans_O(q)$.

1. $Ans \leftarrow \{(r_q, p : a, v_q) \mid a_q \preceq a\}$;
2. **for all** p' such that $\exists Q^1, \dots, Q^k$ p' -paths from r_q to v_q **do**
3. $B \leftarrow \{b \in \mathcal{A} \mid \exists i \in [1, k] \text{ s.t. } \forall a' \in A_{Q^i}, b \preceq a'\}$;
4. $C \leftarrow \{c \in \mathcal{A} \mid \forall b \in B, b \preceq c\}$;
5. $D \leftarrow \{d \in C \mid \nexists d' \in C, d' \neq d \text{ s.t. } d' \preceq d\}$;
6. $Ans \leftarrow Ans \cup \{(r_q, p' : d, v_q) \mid d \in D \wedge a_q \preceq d\}$;
7. **end**
8. **return** $\{(r', p' : a', v') \in Ans \mid \nexists S \subseteq Ans - \{(r', p' : a', v')\} \text{ s.t. } S \models (r', p' : a', v')\}$;

Fig. 5. Answering atomic aRDF queries $(r_q, ?p : a_q, v_q)$

An even tighter complexity bound holds when the annotation is a complete lattice. In this case, after computing the set A on line 11, we can simply compute the least upper bound of the elements in A and thus obtain set C (on line 13). For complete lattices such as $\mathcal{A}_{time-int}$, this can be done in at most a linear number of steps in $|A|$. Thus, the overall complexity of the algorithm becomes $\mathcal{O}(n^2 \cdot e + n \cdot e \cdot a)$.

Algorithm *atomicAnswerP* given in Figure 5 computes the answer to atomic queries with an unknown property. The main difference from *atomicAnswerV* is that the graph we need to explore is the one containing all paths between r and v , instead of the one containing all p -paths starting at r . Depending on the shape of the aRDF theory (e.g., breadth vs. depth of the aRDF graph), either search space may be larger, but the worst case complexity is identical.

PROPOSITION 5.3. *Let O be an aRDF theory graph and let n be the number of vertices in the theory graph O , let $e = |O|$ and let $p = |\mathcal{P}|$. Let (\mathcal{A}, \preceq) be a partial order and let $a = |\mathcal{A}|$. Then the following hold:*

- (1) *atomicAnswerP($O, \mathcal{A}, \preceq, q$) returns $Ans_O(q)$.*
- (2) *atomicAnswerP($O, \mathcal{A}, \preceq, q$) is $\mathcal{O}(n^2 \cdot e + n \cdot e \cdot a^2)$.*

Proof. The correctness of the algorithm again follows from Theorem 4.5. The case of non-transitive properties is handled directly in the initialization of Ans on line 1. Similarly to algorithm *atomicAnswerV*, lines 2–7 handle case (E2) of Theorem 4.5.

The complexity follows from the loop in lines 2–7 of the algorithm. Computing all paths between r_q and v_q takes $\mathcal{O}(n^2 \cdot e)$ iterations, and there are at most $\mathcal{O}(n)$ paths (in the worst case, a path that passes through every node in the theory different from r_q and v_q). Each path has less than $\mathcal{O}(e)$ edges, hence we obtain the same complexity result as for *atomicAnswerV*.

Algorithm *atomicAnswerA* given in Figure 6 computes the answer to atomic queries with unknown annotation. For *atomicAnswerA*, r, p, v are all known therefore the algorithm has lower complexity than its two counterparts.

PROPOSITION 5.4. *Let O be an aRDF theory graph and let n be the number of vertices in the theory graph O , let $e = |O|$ and let $p = |\mathcal{P}|$. Let (\mathcal{A}, \preceq) be a partial order and let $a = |\mathcal{A}|$. Then the following hold:*

- (1) *atomicAnswerA($O, \mathcal{A}, \preceq, q$) returns $Ans_O(q)$.*
- (2) *atomicAnswerA($O, \mathcal{A}, \preceq, q$) is $\mathcal{O}(n \cdot e \cdot a^2)$.*

Proof. The correctness of the algorithm also follows from Theorem 4.5 - the two branches of the conditional on line 3 correspond to cases (E1) and (E2) respectively. Since we now know the resource, property and value in the query, the step in which we compute all paths (line 11) can be performed in at most $\mathcal{O}(n \cdot e)$ steps. This means that, similarly to the previous two atomic answer algorithms, the complexity of *atomicAnswerA* is $\mathcal{O}(n \cdot e + n \cdot e \cdot a^2) = \mathcal{O}(n \cdot e \cdot a^2)$.

5.2 Simple non-atomic queries

In the previous section, we have defined algorithms that compute the answer to atomic queries in polynomial time, by avoiding the expensive computation of $\text{lfp}(O)$. The common trait of all *atomicAnswerX* (where X is one of S, V, P, A) is that

Algorithm atomicAnswerA($O, \mathcal{A}, \preceq, q$)**Input:** Consistent aRDF theory O , annotation (\mathcal{A}, \preceq) and query $q = (r_q, p_q : ?a, v_q)$.**Output:** $Ans_O(q)$.**Notation:** For a property p we write $SP(p) = \{q \in \mathcal{P} \mid (q, rdfs : subPropertyOf^*, p)\}$. We denote by $O|_p$ the restriction of the aRDF graph O to triples labeled with properties in $SP(p)$.

1. $O \leftarrow O|_{p_q}$;
2. $Ans \leftarrow \emptyset$;
3. **if** p_q is non-transitive **then**
4. **for** $(r_q, p', v_q) \in \{(r_q, p' : a', v_q) \in O \mid p' \in SP(p_q)\}$ **do**
5. $A \leftarrow \{a' \in \mathcal{A} \mid (r_q, p' : a', v_q) \in O\}$;
6. $B \leftarrow \{b \in \mathcal{A} \mid \forall a \in A, a \preceq b\}$;
7. $C \leftarrow \{c \in B \mid \nexists c' \in B, c' \neq c \text{ s.t. } c' \preceq c\}$;
8. $Ans \leftarrow Ans \cup \{(r_q, p' : c, v_q) \mid c \in C\}$;
9. **end**
10. **else if** p_q transitive **then**
11. $\{Q^1, \dots, Q^k\} \leftarrow \{p\text{-paths from } r_q \text{ to } v_q\}$;
12. $B \leftarrow \{b \in \mathcal{A} \mid \exists i \in [1, k] \text{ s.t. } \forall a' \in A_{Q^i}, b \preceq a'\}$;
13. $C \leftarrow \{c \in \mathcal{A} \mid \forall b \in B, b \preceq c\}$;
14. $D \leftarrow \{d \in C \mid \nexists d' \in C, d' \neq d \text{ s.t. } d' \preceq d\}$;
15. $Ans \leftarrow Ans \cup \{(r_q, p_q : d, v_q) \mid d \in D\}$;
16. **end**
17. **return** Ans ;

Fig. 6. Answering atomic aRDF queries $(r_q, p_q : ?a, v_q)$

they compute a part of $\text{lfp}(O)$ localized to the subset of the theory that contains the answer to the query. We extend this approach to simple non-atomic queries.

Algorithm answerSV($O, \mathcal{A}, \preceq, q$)**Input:** Consistent aRDF theory O , annotation (\mathcal{A}, \preceq) and query $q = (?r, p_q : a_q, ?v)$.**Output:** $Ans_O(q)$.**Notation:** For a property p we write $SP(p) = \{q \in \mathcal{P} \mid (q, rdfs : subPropertyOf^*, p)\}$. We denote by $O|_p$ the restriction of the aRDF graph O to triples labeled with properties in $SP(p)$.

1. $O \leftarrow O|_{p_q}$;
2. $Ans \leftarrow \emptyset$;
3. **if** p_q is non-transitive **then**
4. **for** $(r', p', v') \in \{(r', p' : a, v') \in O\}$ **do**
5. $A \leftarrow \{a \in \mathcal{A} \mid (r', p' : a, v') \in O\}$;
6. $B \leftarrow \{b \in \mathcal{A} \mid \forall a \in A, a \preceq b\}$;
7. $C \leftarrow \{c \in B \mid \nexists c' \in B, c' \neq c \text{ s.t. } c' \preceq c\}$;
8. $Ans \leftarrow Ans \cup \{(r', p' : c, v') \mid c \in C \wedge a_q \preceq c\}$;
9. **end**
10. **else if** p_q transitive **then**
11. **for all** r', v' s.t. $\exists Q^1, \dots, Q^k$ p_q -paths from r' to v' **do**
12. $B \leftarrow \{b \in \mathcal{A} \mid \exists i \in [1, k] \text{ s.t. } \forall a' \in A_{Q^i}, b \preceq a'\}$;
13. $C \leftarrow \{c \in \mathcal{A} \mid \forall b \in B, b \preceq c\}$;
14. $D \leftarrow \{d \in C \mid \nexists d' \in C, d' \neq d \text{ s.t. } d' \preceq d\}$;
15. $Ans \leftarrow Ans \cup \{(r', p_q : d, v') \mid d \in D \wedge a_q \preceq d\}$;
16. **end**
17. **end**
18. **return** Ans ;

Fig. 7. Answering simple aRDF query $(?r, p_q : a_q, ?v)$

As an example, we show an algorithm for a simple non-atomic query in which the subject and value are both variables, i.e., a simple query of the form $(?r, p : a, ?v)$. The algorithm in Figure 7 is an adaptation of *atomicAnswerS* and *atomicAnswerV* to answer such queries. There are two main differences between *answerSV* and *atomicAnswerV*:

- (1) In *answerSV*, line 4 now iterates through all the (r, p, v) combinations in $O|_{p_q}$, whereas in *atomicAnswerV* it would iterate through all the (r_q, p, v) combinations since r_q was a constant.
- (2) In *answerSV*, line 11 iterates through all the pairs r', v' that have a p_q -path between them, whereas in *atomicAnswerV* it only iterates through the nodes v' that r_q has a p_q -path to.

PROPOSITION 5.5. *Let O be an aRDF theory graph and let n be the number of vertices in the theory graph O , let $e = |O|$ and let $p = |\mathcal{P}|$. Let (\mathcal{A}, \preceq) be a partial order and let $a = |\mathcal{A}|$. Then the following hold:*

- (1) *answerSV($O, \mathcal{A}, \preceq, q$) returns $Ans_O(q)$.*
- (2) *answerSV($O, \mathcal{A}, \preceq, q$) is $\mathcal{O}(n^3 \cdot e + n^2 \cdot e \cdot a^2)$.*

Proof The correctness of the algorithm follows from Theorem 4.5. Similarly to the proof of Proposition 5.2, in lines 4–9 we include in the answer all aRDF triples on the property p_q that match condition (E1) of Theorem 4.5 and on lines 10–16 we analyze all aRDF p_q paths in the aRDF theory that match condition (E2) of Theorem 4.5. As seen before, all triples entailed by the aRDF theory fall into one of the above-mentioned categories, therefore the algorithm returns all possible answers.

In terms of complexity, note that we need $(O)(n^3 \cdot e)$ steps to compute the paths on line 11. There are now at most n^2 p_q -paths considered on line 11, which means line 12 will be run at most $\mathcal{O}(n^2 \cdot e \cdot a^2)$. The overall complexity of the algorithm will therefore be $\mathcal{O}(n^3 \cdot e + n^2 \cdot e \cdot a^2)$. Note that a complexity of this form is intuitively what we expect, since the difference between *atomicAnswerV* and *answerSV* is that the subject of the query becomes variable. Therefore, we can obtain *answerSV* from *atomicAnswerV* by adding an enclosing loop that iterates through all possible values for $?r$. Indeed, the complexity for *answerSV* can be computed by multiplying the complexity value of *atomicAnswerV* by the number of resources n . The same process can be applied to answer any simple non-atomic query. Since the algorithms are very similar to their *atomicAnswer* counterparts, we omit their formal descriptions.

5.3 Conjunctive queries

For simple queries, we clearly want to avoid the expensive computation of $\text{lfp}(O)$ in *naiveSimpleAnswer*. For conjunctive queries, it is not directly clear which of computing $\text{lfp}(O)$ or computing the cartesian product $\prod_{q \in Q} \Theta_Q(q)$ in step (2) of *naiveConjunctiveAnswer* is more computationally intensive. The comparison depends on both the aRDF theory O and on the “size” of the conjunctive query Q . Therefore, we propose two distinct methods of answering conjunctive queries:

- (1) *conjunctAnswer_GraphMatching* is based on the observation that conjunctive queries are partially instantiated aRDF graphs. Therefore, inexact graph

matching algorithms [Hlaoui and Wang 2002; Cordella et al. 2004] can be used to match the query graph against the aRDF theory graph. To obtain a correct answer, graph matching must be performed on the closure of the original theory — therefore we must compute $\text{lfp}(O)$.

- (2) *conjunctAnswer_Ordering* uses the efficient simple query answering algorithms to derive answers for the elements of the conjunctive query, thus avoiding the fixpoint computation. The algorithm uses a heuristic ordering of the elements in the conjunctive query to compute the smallest possible part of the cartesian product.

Algorithm *conjunctAnswer_GraphMatching* (Figure 8) starts by computing the closure $\text{lfp}(O)$ on line 1. After $\text{lfp}(O)$ is computed, inexact graph matchings [Hlaoui and Wang 2002; Cordella et al. 2004] are used to determine potential answers to the conjunctive query (line 3). Since graph matching algorithms cannot take the semantics of the aRDF annotations into account, we have to check the potential answer triple by triple against the query annotation (if constant) on lines 7–11. If all triples have “better” annotations (in terms of the \preceq order) than the corresponding query triples, the answer is stored (line 14). The worst-case complexity of *conjunctAnswer_GraphMatching* is $\mathcal{O}(n!)$ in the worst case, since graph matching algorithms are factorial in the size of the graph [Cordella et al. 2004]. However, we have determined experimentally (Section 7) that the average complexity is close to polynomial in the size of the theory and in the size of the query.

Algorithm *conjunctAnswer_GraphMatching*($O, \mathcal{A}, \preceq, Q$)

Input: Consistent aRDF theory O , annotation (\mathcal{A}, \preceq) and query $Q = \{q_i = (r_i, p_i : a_i, v_i) | i \in [1, m]\}$.

Output: $\text{Ans}_O(Q)$.

1. $O \leftarrow \text{lfp}(O)$;
2. $\text{Ans} \leftarrow \emptyset$;
3. **execute** graph matching algorithm on G_Q and O ;
4. **for** all matchings between G_Q and O **do**
5. $ok \leftarrow true$;
6. **for** $i \in [1, m]$ **do**
7. $(r, p : a, v) \leftarrow$ the triple in O matched to q_i ;
8. **if** $\neg(a_i \text{ variable}) \wedge \neg(a_i \preceq a)$ **then**
9. $ok \leftarrow false$;
10. **break**;
11. **end**
12. **end**
13. **if** ok **then**
14. $\text{Ans} \leftarrow \text{Ans} \cup \{\text{set of triples matched to } G_Q\}$;
15. **end**
16. **return** Ans ;

Fig. 8. Answering conjunctive aRDF queries through inexact graph matching

Algorithm *conjunctAnswer_Ordering* (Figure 10) starts by creating a partial order of the component queries in the conjunctive query Q (lines 1–6). The partial order indicates which parts of the cartesian product should be computed first in order to minimize the number of operations. The process is similar to that of

determining the ordering of joins in a relational database. To create a partial order for the component queries, we create an undirected labeled graph H_Q (line 1) as follows:

- Each $q_i \in Q$ is a vertex in H_Q . There are no other vertices in H_Q .
- There exists an edge between q_i and q_j labeled with $?v$ iff there exists a variable $?v$ that appears in both q_i and q_j .

The resulting H_Q graph may contain cycles for certain queries. For instance, the conjunctive query $Q = \{q_1 = (?v1, associatedWith : .65, ?v2), q_2 = (?v2, associatedWith : .4, ?v1)\}$ results in a graph with two edges, both between q_1 and q_2 , labeled with $?v1$ and $?v2$ respectively. In such cases, it is not clear which of q_1, q_2 should be executed first. To break cycles, we use an estimate of the cardinality of each component query based on a very recent method [Maduko et al. 2007] that uses a pattern-based summarization framework to estimate the cardinality of RDF graph patterns. The method uses minimal overhead, especially since we only estimate cardinalities for simple queries. Based on the cardinality estimation, we break cycles as follows:

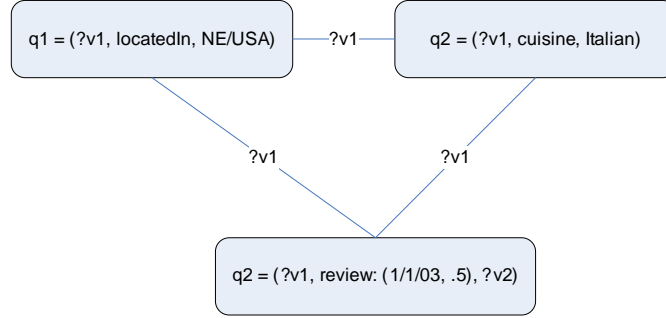
- (1) For each cycle, we choose the node $q_i \in H_Q$ with the lowest cardinality.
- (2) Let q_j be the neighbor of q_i with the highest estimated cardinality.
- (3) We remove the edges between q_i and q_j .

Intuitively, we remove those edges (lines 2—6) that would not allow a query with a low estimated cardinality to be executed before other queries with higher cardinality estimates. The cycle-free graph thus obtained is a partial order of the component queries that minimizes the number of cartesian product operations.

Finally, the algorithm takes the depth-first traversal of H_Q (line 8) and incrementally computes the cartesian product of the sets of substitutions for each component query (lines 12—14). We use a depth-first traversal rather than a breadth-first traversal to minimize the number of operations. A breadth-first traversal will compute cartesian products between the $\Theta_O(q_i)$ and $\Theta_O(q_j)$, even if q_i and q_j have no variables in common. This will produce a result of size $|\Theta_O(q_i)| \cdot |\Theta_O(q_j)|$ (since there are no common variables). However, if we go depth-first, we will favor queries q_i, q_j with common variables and thus with a smaller size for the cartesian product $\Theta_O(q_i) \times \Theta_O(q_j)$.

We will illustrate the *conjunctAnswer_Ordering* through an example.

EXAMPLE 5.6. Consider the query in Figure 3 on the *aRDF* theory in Figure 1(c). The H_Q graph for this conjunctive query is shown in Figure 9. Since there is a cycle, the algorithm will enter the loop on line 2. In this case, the cardinality estimation method will most likely give lower cardinalities to q_1 and q_2 , which contain a single variable. Let us assume that q_2 has the lowest cardinality and q_3 the highest. On line 5, the edge between q_2 and q_3 will be deleted and the depth-first traversal on line 8 will yield $L = \{q_2, q_1, q_3\}$. On line 11, we will remove q_2 from the queue and compute (line 12) $\Theta_O(q_2) = \{?v1 \leftarrow \text{Charlie's}, ?v1 \leftarrow \text{Grivanti}\}$. $\Theta = \Theta_O(q_2)$. At the next step, we will remove q_1 from the queue and obtain the same set of substitutions, which means Θ will not change. Finally, on the third iteration, $\Theta_O(q_3) = \{(?v1 \leftarrow \text{Charlie's}, ?v2 \leftarrow \text{Reviewer \#21765}), (?v1 \leftarrow \text{Grivanti}, ?v2 \leftarrow$

Fig. 9. Example H_Q graph

Algorithm conjunctAnswer_Ordering($O, \mathcal{A}, \preceq, Q$)

Input: Consistent aRDF theory O , annotation (\mathcal{A}, \preceq) and query $Q = \{q_i = (r_i, p_i : a_i, v_i) | i \in [1, m]\}$.

Output: $Ans_O(Q)$.

1. **construct** graph H_Q ;
2. **while** there exists a cycle in H_Q **do**
3. **choose** q_i with the lowest $card(q_i)$ in the cycle;
4. $q_j \leftarrow \text{argmax}_{card(q)}(\{q | \exists(q, q_i) \in H_Q\})$;
5. $H_Q \leftarrow H_Q - \{(q_i, q_j)\}$;
6. **end**
8. $L \leftarrow$ depth-first traversal of H_Q starting with the lowest cardinality;
9. $\Theta \leftarrow \emptyset$
10. **while** $L \neq \emptyset$ **do**
11. $q \leftarrow \text{dequeue}(L)$;
12. **compute** $\Theta_O(q)$;
13. $\Theta \leftarrow \Theta \times \Theta_O(q)$;
14. $\Theta \leftarrow \Theta - \{(\theta_1, \dots, \theta_k) | \theta_1 \cup \dots \cup \theta_k \text{ inconsistent}\}$;
15. **end**
16. **compute** $A_O(Q)$ and $Ans_O(Q)$ based on Θ ;
17. **return** Ans ;

Fig. 10. Answering conjunctive aRDF by heuristic ordering of the component queries

Reviewer #21765), $(?v1 \leftarrow \text{Charlie's}, ?v2 \leftarrow \text{Reviewer #16742})$. After computing the cartesian product and removing inconsistent substitutions, $\Theta = \Theta_O(q_3)$.

PROPOSITION 5.7 CORRECTNESS. *The algorithm conjunctAnswer_Ordering($O, \mathcal{A}, \preceq, Q$) is correct, i.e., it terminates and returns $Ans_O(Q)$.*

Proof. The loop on lines 2–6 will always terminate since we are removing at least one edge from H_Q at each iteration. This implies that eventually H_Q will be cycle free. The loop on line 10–15 is on the finite set L . Note that lines 10–16 correspond to the process described in Definition 4.12. Since we are computing a cartesian product, the particular order that we choose in lines 1–6 does not affect the correctness of the result, only the computation time.

PROPOSITION 5.8 COMPLEXITY. *The worst time complexity of conjunctAnswer_Ordering is $\mathcal{O}((n^2 \cdot p)^{|Q|})$, where $n = |\mathcal{R}|$, $p = |\mathcal{P}|$ and $|Q|$ is the number of simple queries in*

Q .

Proof. For each component query, the worst case cardinality of the answer is $\mathcal{O}(n^2 \cdot p)$. Since we have p such simple queries, the worst-case cartesian product is $\mathcal{O}((n^2 \cdot p)^{|Q|})$. However, our experimental results show that in the average case, the size of the result will be far below the worst-case limit.

6. ARDF VIEW MAINTENANCE

In this section, we explore solutions to the **aRDF view maintenance** problem. Suppose a query q is often posed by users: it then becomes efficient to store the results of q and if possible avoid the expensive re-computation of $Ans_O(q)$ by incrementally updating the result when the underlying **aRDF** theory changes. Views are omnipresent in databases and there is a large literature on them summarized in [Gupta and Mumick 1999]. The queries q defining a view can be used to express conditions that users want to track.

In order to maintain **aRDF** views, we require an additional data structure called the *path annotation function* be stored with the **aRDF** theory. This new data structure exploits the fact that, as seen in the atomic answer algorithms, we are interested mainly in the annotations on a p -path rather than the actual nodes the path goes through. In the following sections, we will explore in detail how the *path annotation function* can be computed, as well as incrementally checking consistency when the **aRDF** theory changes and maintaining result sets for queries for insert, delete and update operations. For the rest of the section, let $q = (r_q, p_q : a_q, v_q)$ be a simple query and let $R = Ans_O(q)$ be the store answer to q . We will present our view maintenance algorithms for simple queries only for reasons of simplicity. Conjunctive queries can be easily maintained by repeatedly applying the maintenance algorithms for each component of the conjunction.

There are two main challenges in incrementally updating views:

- (1) Check the **aRDF** theory consistency incrementally when the theory changes through insertions and/or deletions.
- (2) Re-compute the answers to the queries incrementally.

6.1 Path Annotation Function

In order to recompute path annotations quickly when the **aRDF** theory changes, we need to maintain an additional data structure called the *path annotation function*. We point out that in all the *atomicAnswer* and *answer* algorithms, we are only interested in the sets of annotations on each path and not the actual resources on the path. Therefore, we only need store subsets of \mathcal{A} that annotate p -paths in the **aRDF** theory to quickly re-compute query answers when the theory changes.

Definition 6.1 Path annotation function. Let q be a simple query. The path annotation function δ for q is a function $\delta : Ans_O(q) \rightarrow 2^{2^{\mathcal{A}}}$ such that $\forall (r, p : a, v) \in R, P$ is a p -path between r and v iff $A_P \in \delta(r, p : a, v)$.

In short, a path annotation function maps elements of an answer set to a query to a set containing sets of annotations. Each element $(r, p : a, v)$ is mapped to a set X in which every element $A \in X$ is the set of annotations for a p -path between r and v .

EXAMPLE 6.2. Consider the *aRDF* theory in Figure 1(b) and the query $q=(Flu, associatedWith: .5, ?v)$. The answer to this query is $Ans_O(q) = \{(Flu, associatedWith: .65, Pneumonia), (Flu, hasComplication: .7, AcuteBronchitis)\}$ and $\delta(Flu, associatedWith: .65, Pneumonia) = \{(.7, .65), (.15)\}$.

```

.....
10. else if  $p$  transitive then
11.   for all  $v'$  s.t.  $\exists Q^1, \dots, Q^k$   $p$ -paths from  $r$  to  $v'$  do
12'.    $\delta(r, p, v') \leftarrow \{A_{Q^1}, \dots, A_{Q^k}\};$ 
12.    $B \leftarrow \{b \in \mathcal{A} \mid \exists i \in [1, k] \text{ s.t. } \forall a' \in A_{Q^i}, b \preceq a'\};$ 
.....

```

Fig. 11. Computing the path annotation function

We should note a few important properties of the path annotation function:

- (1) $\delta(r, p : a, v)$ does not depend on the annotation a ; more precisely, $\delta(r, p : a, v) = \delta(r, p : a', v), \forall (r, p : a, v), (r, p : a', v) \in Ans_O(q)$. We will often abuse notation and write simply $\delta(r, p, v)$ to denote the value of the path annotation function.
- (2) δ is a shared data structure. In other words, δ is not dependent on a particular query q . The path annotation function can be computed and stored either at system startup or incrementally, as queries are being answered.

As an example of how to compute δ incrementally, Figure 11 shows how to do this during the *atomicAnswerV* algorithm. In the newly inserted line (12'), the value of δ for a triple that could be in the answer can be simply stored from what the algorithm has already computed.

6.2 Incremental Consistency Checking

In this section, we look at the problem of incremental consistency verification and answer re-computation when a new triple is inserted into the *aRDF* theory. Let $(r_i, p_i : a_i, v_i)$ be the newly inserted triple. Of course, our goal is to avoid a full re-computation if possible.

Although the *aRDFconsistency* algorithm is quite efficient in practice, we wonder whether we can do better by analyzing only a part of the *aRDF* theory which is “close” to the newly inserted triple.

EXAMPLE 6.3. Consider the example *aRDF* theory in Figure 1(b) and let the triple to be inserted be $(Pneumonia, associatedWith: .25, CorPulmonale)$. The algorithm will determine on line 6 that *Flu* and *CorPulmonale*, as well as *Pneumonia* and *CorPulmonale* are linked together by new paths. Let us consider the step in which $r = Flu, r' = CorPulmonale$. The path annotations are recomputed in A to be $\{(.7, .001), (.15, .25)\}$. B will be computed to be the interval $[0, .15]$ and the condition on line 10 is clearly false. After verifying the remaining pair of newly connected resources, the algorithm will return *True*.

THEOREM 6.4. Let O be a consistent *aRDF* theory and let $(r_i, p_i : a_i, v_i)$ be an *aRDF* triple. Then *aRDFconsistencyInsert* $(O, (r_i, p_i : a_i, v_i))$ returns true iff $O \cup \{(r_i, p_i : a_i, v_i)\}$ is a consistent *aRDF* theory.

Algorithm *aRDFconsistencyInsert*($O, (r_i, p_i : a_i, v_i)$)

Input: Consistent *aRDF* theory O , newly inserted triple $(r_i, p_i : a_i, v_i)$.

Output: True iff $O \cup \{(r_i, p_i : a_i, v_i)\}$ is consistent.

Notation: For a property p we write $SP(p) = \{q \in \mathcal{P} \mid (q, rdfs : subPropertyOf^*, p)\}$. We denote by $O|_p$ the restriction of the *aRDF* graph O to triples labeled with properties in $SP(p)$. $N(O)$ denotes the set of nodes in the *aRDF* theory graph O .

1. $A \leftarrow \{a \in \mathcal{A} \mid \exists (r_i, p_i : a, v_i) \in O\} \cup \{a_i\}$;
2. **if** $\exists a \in \mathcal{A}$ s.t. $\forall a' \in A, a' \preceq a$ **then return False**;
3. **for** $p \in \mathcal{P}$ transitive **do**
4. $O' \leftarrow O|_p$;
5. $O'' \leftarrow (O \cup \{(r_i, p_i : a_i, v_i)\})|_p$;
6. $P \leftarrow \{\text{paths } Q \subseteq O'' \mid \exists Q' \subseteq O'', Q' \supset Q\}$;
7. **for** r, r' connected by additional paths in O'' than in O' **do**
8. $P' \leftarrow \{Q \in P \mid Q \text{ is a path between } r, r'\}$;
9. $A \leftarrow \{A_Q \mid Q \in P'\}$;
10. $B \leftarrow \{b \in \mathcal{A} \mid \exists A_Q \text{ s.t. } \forall a \in A_Q, b \preceq a\}$;
11. **if** $\exists a \in \mathcal{A}$ s.t. $\forall b \in B, b \preceq a$ **then return False**;
12. **endfor**
13. **return True**;

Fig. 12. Incremental consistency verification for insertions

Justification. Let us assume that p_i is a non-transitive property. According to condition (C1) of Theorem 3.4, $O \cup \{(r_i, p_i : a_i, v_i)\}$ is consistent if and only if the set of annotations on triples (r_i, p_i, v_i) has an upper bound; the lines 1–2 of *aRDFconsistencyInsert* will return *False* if and only if the set does not have such a bound. If p_i is a transitive property and since O is consistent, the only paths that could cause $O \cup \{(r_i, p_i : a_i, v_i)\}$ to be inconsistent according to condition (C2) of Theorem 3.4 are paths newly created by $(r_i, p_i : a_i, v_i)$. *aRDFconsistencyInsert* considers all such paths (lines 6–11) for all transitive properties in the theory (lines 3–7) and returns *False* if and only if such paths do not verify condition (C2) of Theorem 3.4 (lines 7–10).

In the worst case, the complexity of the *aRDFconsistencyInsert* is the same as *aRDFconsistency*; however such a scenario requires that every edge in the theory including $(r_i, p_i : a_i, v_i)$ is on a p -path between the same two nodes. In the general case, *aRDFconsistencyInsert* looks only at the strongly connected component of $O|_{p_i}$ that contains the newly inserted triple.

6.3 View Maintenance on Insertion

We now look at the problem of incrementally computing $R' = Ans_{O \cup \{(r_i, p_i : a_i, v_i)\}}(q)$ from R and δ . The algorithm *viewMaintenanceInsert* show in Figure 13 performs this incremental computation. To keep the formal description as simple as possible, we assume that p_q is a constant; the cases in which p_q is variable are a straightforward extension. We will also assume that δ is updated accordingly after a successful insertion. This can be done while performing the view maintenance on insertion, in the same way as in Figure 11.

The algorithm starts by analyzing the cases in which the property of the triple to be inserted is non-transitive (lines 3–9). If this is the case, we have to recompute

```

method addResult( $Res, A, r, p, v$ )
1.  $B \leftarrow \{b \in A \mid \forall a \in A, a \preceq b\}$ ;
2.  $C \leftarrow \{b \in B \mid \nexists b' \in B, b' \neq b \text{ s.t. } b' \preceq b\}$ ;
3.  $Res \leftarrow Res - \{(r, p : a, v) \in R\}$ ;
4.  $Res \leftarrow Res \cup \{(r, p : c, v) \mid c \in C\}$ ;
end

```

Algorithm viewMaintenanceInsert($O, q, R, \delta, (r_i, p_i : a_i, v_i)$)

Input: Consistent aRDF theory O , query $q = (r_q, p_q : a_q, v_q)$, answer R , precomputed path annotation function δ and newly inserted triple $(r_i, p_i : a_i, v_i)$.

Output: $R' = Ans_{O \cup \{(r_i, p_i : a_i, v_i)\}}(q)$.

Notation: For a property p we write $SP(p) = \{q \in \mathcal{P} \mid (q, rdfs : subPropertyOf^*, p)\}$. We denote by $O|_p$ the restriction of the aRDF graph O to triples labeled with properties in $SP(p)$. $N(O)$ denotes the set of nodes in the aRDF theory graph O .

```

1.  $R' \leftarrow R$ ;
2. if  $p_q$  is transitive and  $p_i \notin SP(p_q)$  then return  $R$ ;
3. if  $p_i$  is not transitive then
4.   if  $\exists a \in A$  s.t.  $(r_i, p_i : a, v_i) \in R$  do
5.      $A \leftarrow \{a' \in A \mid (r, p_i : a', v_i) \in O\} \cup \{a_i\}$ ;
6.      $B \leftarrow \{b \in A \mid \forall a \in A, a \preceq b\}$ ;
7.      $C \leftarrow \{c \in B \mid \nexists c' \in B, c' \neq c \text{ s.t. } c' \preceq c\}$ ;
8.      $R' \leftarrow R - \{(r_i, p_i : a, v_i) \in R\} \cup \{(r_i, p_i : c, v_i) \mid c \in C \wedge a \preceq c\}$ ;
9.   endif
10. else //  $p_i$  is transitive
11.   for  $(r, p : a, v) \in R$  do
12.     for  $p' \in SP(p_i) \cap SP(p)$  do
13.       if  $(r_i, p_q, v)$  semi-unifiable with  $q$  then
14.         for all  $p'$ -paths  $P$  from  $v_i$  to  $r$  do
15.            $A \leftarrow \{b \preceq a_i \mid \exists S \in \delta(r, p, v) \text{ s.t. } \forall a' \in S \cup A_P, b \preceq a'\}$ ;
16.           addResult( $R', A, r_i, p', v$ );
17.         endfor
18.       if  $(r, p_q, v_i)$  semi-unifiable with  $q$  then
19.         for all  $p'$ -paths  $P$  from  $v$  to  $r_i$  do
20.            $A \leftarrow \{b \preceq a_i \mid \exists S \in \delta(r, p, v) \text{ s.t. } \forall a' \in S \cup A_P, b \preceq a'\}$ ;
21.           addResult( $R', A, r, p', v_i$ );
22.         endfor
23.       for all  $p'$ -paths  $P_1$  from  $r$  to  $r_i$  and  $P_2$  from  $v_i$  to  $v$  do
24.          $A \leftarrow \{b \preceq a_i \mid \exists S \in \delta(r, p, v) \text{ s.t. } \forall a' \in S \cup A_{P_1} \cup A_{P_2}, b \preceq a'\}$ ;
25.         addResult( $R', A, r, p', v$ );
26.       endfor
27.     endfor
28.   endfor
29.   for  $r, v \in N(O) \times N(O)$  such that  $(r, p_q, v)$  semi-unifiable with  $q$  do
30.     for  $p' \in SP(p_i) \cap SP(p)$  do
31.       for all  $p'$ -paths  $P_1$  from  $r$  to  $r_i$  and  $P_2$  from  $v_i$  to  $v$  do
32.          $A \leftarrow \{a \preceq a_i \mid \forall a' \in A_{P_1} \cup A_{P_2}, a \preceq a'\}$ ;
33.         addResult( $R', A, r, p', v$ );
34.       endfor
35.     endfor
36.   endfor
37. endif
38. return  $R'$ ;

```

Fig. 13. View maintenance for atomic queries for insertions
ACM Transactions on Computational Logic, Vol. V, No. N, December 2007.

the sets of annotation on direct edges between r_i and v_i on the property p_i . If p_i is transitive, then we analyze three different cases for each element in the previous query answer R :

- (1) If r_i has become connected through p_q paths to nodes $v \neq v_i$ (lines 13–17)
- (2) If v_i has become connected through p_q paths to nodes $r \neq r_i$ (lines 18–22)
- (3) If the new edge between r_i and v_i creates a new p_q path from r to v through r_i and v_i – in which case r and v will become p_q -connected (lines 23–26).

We then recompute the annotations for the affected (or new) paths and update the result accordingly. So far, we have only analyzed updates to elements in the answer. In lines 29 – 37 we also analyze whether any new triples should be added to the answer (similar to case (3) above, but for resources r and v that do not belong to a triple $(r, p, v) \in R$).

EXAMPLE 6.5. Consider the example *aRDF* theory in Figure 1(b) and let the triple to be inserted be $(Pneumonia, associatedWith: .25, CorPulmonale)$. The query to be maintained is $q=(Flu, associatedWith: .15, ?v)$ and the answer **before insertion** is $Ans_O(q)=\{(Flu, associatedWith : .65, Pneumonia), (Flu, hasComplication: .7, AcuteBronchitis)\}$. Since *associatedWith* is transitive, the algorithm will follow the branch starting on line 10. R will not change until we reach line 23, where we find a new path linking *Flu* and *CorPulomonale* through *Pneumonia*. On line 24, $A = [0, .15]$ and we will add a new triple to the result: $(Flu, associatedWith: .15, CorPulmonale)$.

THEOREM 6.6. Let O be a consistent *aRDF* theory and let $(r_i, p_i : a_i, v_i)$ be an *aRDF* triple such that $O \cup \{(r_i, p_i : a_i, v_i)\}$ is consistent. If q is a simple query and δ is the path annotation function, then $viewMaintenanceInsert(O, q, Ans_O(q), \delta, (r_i, p_i : a_i, v_i))$ returns $Ans_{O \cup \{(r_i, p_i : a_i, v_i)\}}(q)$.

Proof. Let us assume that *viewMaintenanceInsert* does not return the correct answer. This means we are in one of two cases: (i) either the algorithm returns a triple $(r, p : a, v)$ that is not an answer to the query or (ii) there exists an answer to the query that is not in the answer returned by the algorithm. We will examine each case in turn.

Let us assume that there exists a triple $(r, p : a, v)$ returned by the algorithm that is not an answer to q . $(r, p : a, v)$ may not be an answer to q for two reasons.

- (1) $(r, p : a, v)$ is not semi-unifiable with q . For non-transitive properties p , note that we are only returning triples $(r_i, p_i : c, v_i)$ on line 8 such that $(r_i, p_i : a, v_i)$ was already in the previous answer R (line 4), hence it was semi-unifiable with the query. For transitive properties, we only analyze triples that are semi-unifiable with q (lines 13, 18, 23, 29).
- (2) For queries with a constant annotation a_q , we have that $a_q \not\leq a$. That cannot be the case due to the conditions imposed on any triples added to the result on line 8 and through *addResult* on lines 15–16, 20–21, 24–25 and 32–33.

We have established that we cannot have a triple returned by the algorithm that is not an answer to the query. Let us assume that there exists an answer to the query q that will not be returned. Since we do not change parts of the answer that

are not affected by the inserted triple, the answer we are missing must be related to $(r_i, p_i : a_i, v_i)$. There are several cases in which the inserted triple can affect the answer:

- (1) For non-transitive properties, it may either add a new element to the answer which is identical to the inserted triple or it may alter the annotation for an existing answer with the resource, property and value (r_i, p_i, v_i) . Both cases are handled in lines 4–9.
- (2) For transitive properties, $(r_i, p_i : a_i, v_i)$ can alter the existing paths in the answer R that are semi-unifiable with the query by pre-pending existing paths (handled in lines 14–17), appending to existing paths (lines 19 – 22) or simply connecting two existing portions of a path that were previously not connected (handled in lines 23–26). Finally, the newly inserted triple can create new paths that were not represented by any result in R (handled on lines 29–36).

We point out that the complexity of the view maintenance algorithms is the same as that of the corresponding query algorithms since in the worst case, all triples in the answer may be changed. If this happens, for all practical purposes view maintenance will rerun the query algorithm to recompute all answers. However, we show experimentally that in most cases, performing view maintenance is much faster than recomputing the entire query answer from scratch.

6.4 View maintenance on deletions

Suppose now that we intend to delete the triple $(r_d, p_d : a_d, v_d)$ from O . We would first like to show that deletions do not affect the consistency of an aRDF theory.

THEOREM 6.7. *Let O be a consistent aRDF theory and let $(r, p : a, v) \in O$ be an arbitrary triple. Then $O - \{(r, p : a, v)\}$ is aRDF consistent.*

Proof. Let I be a satisfying interpretation for O . We can easily prove that I satisfies O' :

- (i) I satisfies every triple in O implies that I satisfies any triple in $O' = O - \{(r, p : a, v)\}$.
- (ii) For all transitive properties $p \in \mathcal{P}$ let P be the set of p -paths $Q = \{t_1, \dots, t_k\}$ in O . The set of p -paths in O' is clearly a subset of the set of paths in O . We know for all $a \in \mathcal{A}$ such that $a \preceq a_i$ for all $1 \leq i \leq k$, it is the case that $a \preceq I(r_1, p, r_{k+1})$. That will clearly hold for a subset of the paths considered for O , hence it will hold for O' .

O' has a satisfying interpretation and is thus consistent.

We present an algorithm for computing $R' = Ans_{O - \{(r_d, p_d : a_d, v_d)\}}(q)$ in Figure 14. We again assume that δ is updated accordingly after the deletion. The algorithm starts with a similar procedure as *viewMaintenanceInsert* for non-transitive properties (lines 3–9). For transitive properties, we simply look for p_q -paths in the answer that have been interrupted by the deletion (lines 10–13). We compute the new path annotations by removing from delta the annotations for all the interrupted paths (line 13) and recompute the values for the remaining path annotations (lines 14–16).

Algorithm viewMaintenanceDelete($O, q, R, \delta, (r_d, p_d : a_d, v_d)$)

Input: Consistent aRDF theory O , query $q = (r_q, p_q : a_q, v_q)$, answer R , precomputed function δ and deleted triple $(r_d, p_d : a_d, v_d)$.

Output: $R' = \text{Ans}_{O \cup \{(r_d, p_d : a_d, v_d)\}}(q)$.

Notation: For a property p we write $SP(p) = \{q \in \mathcal{P} \mid (q, rdfs : \text{subPropertyOf}^*, p)\}$. We denote by $O|_p$ the restriction of the aRDF graph O to triples labeled with properties in $SP(p)$. $N(O)$ denotes the set of nodes in the aRDF theory graph O .

1. $R' \leftarrow R$;
2. **if** p_d is not transitive **then**
3. **if** $\exists a \in \mathcal{A}$ s.t. $(r_d, p_d : a, v_d) \in R$ **do**
4. $A \leftarrow \{a' \in \mathcal{A} \mid (r, p' : a', v') \in O\} \cup \{a_i\}$;
5. $B \leftarrow \{b \in \mathcal{A} \mid \forall a \in A, a \preceq b\}$;
6. $C \leftarrow \{c \in B \mid \nexists c' \in B, c' \neq c \text{ s.t. } c' \preceq c\}$;
7. $R' \leftarrow R' - \{(r_d, p_d : a, v_d) \in R\} \cup \{(r, p' : c, v') \mid c \in C \wedge a \preceq c\}$;
8. **endif**
9. **else** // p_d is transitive
10. **for** $(r, p : a, v) \in R$ **do**
11. **if** $\exists S \in \delta(r, p, v)$ s.t. $p_d \in S$ **then**
12. **for** P_1, P_2 p -paths between r, r_d and v_d, v respectively **do**
13. $T \leftarrow \delta(r, p : a, v) - \{A_{P_1} \cup A_{P_2} \cup \{p_d\}\}$;
14. $A \leftarrow \{a' \in \mathcal{A} \mid \exists S \in T \text{ s.t. } \forall a'' \in S, a' \preceq a''\}$;
15. $B \leftarrow \{b \in \mathcal{A} \mid \forall a' \in A, a' \preceq b\}$;
16. $C \leftarrow \{c \in B \mid (\nexists c' \in B, c' \neq c \text{ s.t. } c' \preceq c) \wedge ((a_q \preceq c) \vee (a_q \text{ variable}))\}$;
17. $R' \leftarrow R' - \{(r, p : a, v)\} \cup \{(r, p : c, v) \mid c \in C\}$;
18. **endfor**
19. **endfor**
20. **endif**
21. **return** R' ;

Fig. 14. View maintenance for atomic queries for deletions

EXAMPLE 6.8. Consider the example aRDF theory in Figure 1(b) and let the triple to be deleted be $(AcuteBronchitis, associatedWith : .65, Pneumonia)$. The query to be maintained is $q = (Flu, associatedWith : .25, ?v)$ and the answer **before insertion** is $\text{Ans}_O(q) = \{(Flu, associatedWith : .65, Pneumonia), (Flu, hasComplication : .7, AcuteBronchitis)\}$. Since *associatedWith* is transitive, the algorithm will follow the branch starting on line 9. We find that one of the paths between *Flu* and *Pneumonia* was interrupted, hence $T = \{\{.15\}\}$ on line 13. We recompute $A = [0, .15]$, $B = [.15, 1]$ and $C = \emptyset$. As a result, the triple $(Flu, associatedWith : .65, Pneumonia)$ will be removed from the answer.

THEOREM 6.9. Let O be a consistent aRDF theory and let $(r_d, p_d : a_d, v_d)$ be an aRDF triple. If q is a simple query and δ is the path annotation function, then $\text{viewMaintenanceDelete}(O, q, \text{Ans}_O(q), \delta, (r_d, p_d : a_d, v_d))$ returns $\text{Ans}_{O - \{(r_i, p_i : a_i, v_i)\}}(q)$.

Proof. Let us assume that *viewMaintenanceDelete* does not return the correct answer. This means that either (i) the algorithm returns a triple that is not an answer to q or (ii) there exists a triple that is an answer to q that is not returned.

Let $(r, p : a, v)$ be a triple returned by the algorithm that is not an answer to q . Then we are in one of the following cases:

- (1) $(r, p : a, v)$ is not semi-unifiable with q . Note that due to the conditions on

lines 3 and 12, the triples we are adding to the answer will have a resource, property and value that are already in a triple in R . Therefore, $(r, p : a, v)$ must be semi-unifiable with q .

- (2) If a_q is constant, $a_q \not\leq a$. This cannot be the case due to the annotation recomputation in lines 4–7 and 13–17.

Let us assume that there exists a triple that is an answer to q that is not returned by the algorithm. Clearly, all answers that are unaffected by $(r_d, p_d : a_d, v_d)$ will still be returned. In lines 2–8 we will re-compute the annotation (or remove any answers) if p_d is non-transitive. If p_d is transitive, we re-compute the annotations and remove all answers corresponding to paths “disconnected” by the deleted triple in lines 13–17.

As was the case with *viewMaintenanceInsert*, the complexity of *viewMaintenanceDelete* is also the same as that of the algorithm to compute $Ans_O(q)$ since in the worst case we may have to update the entire answer.

7. EXPERIMENTAL EVALUATION

Our aRDF system implementation consists of approximately 4300 lines of Java code. The experiments were performed on an Intel Pentium 4 3.0 GHz machine with 1GB DDR SDRAM, running openSuse 9.0.2. The aRDF datasets were stored in flat binary files on disk. The running time for all algorithms includes disk I/O.

We experimented on three distinct datasets. The *GovTrack*¹⁴ dataset consists of 20,001,954 RDF triples. Most of the triples have associated temporal information, which marks the duration of validity of the triple. We used the $\mathcal{A}_{time-int}$ annotation based on the attached temporal information to annotate the dataset. This resulted in 12,340,576 aRDF triples.

The *ChefMoz*¹⁵ dataset consists of 802,371 RDF triples, including restaurant review scores and review dates. We used the review information to annotate the dataset with $\mathcal{A}_{fuztime}$. This resulted in 549,781 aRDF triples.

Finally, to study the variations of the query answer time with various features of the aRDF theory, we also generated synthetic datasets ranging from 10,000 to 10,000,000 aRDF triples. For each size, we generated 15 independent random datasets using uniform distributions for the random generator. To make the dataset as close to real-world datasets as possible – based on our study of the previous two and other RDF datasets, we maintained the following characteristics constant during the generation process:

- (1) The number of properties $|\mathcal{P}|$ follows a Gaussian distribution around 0.5% of the size of the dataset, with a standard deviation of no more than 0.01% of the size of the dataset.
- (2) The number of transitive properties was held constant at 5% of the total number of properties.
- (3) The number of *rdfs : subPropertyOf* relations was uniformly distributed between 10 and 20% of the number of properties.

¹⁴<http://www.govtrack.us>.

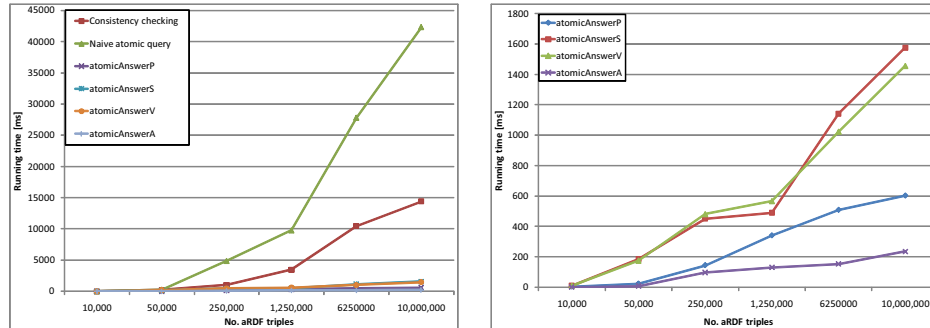
¹⁵<http://chefmoz.org>

Table I. Summary of consistency checking and atomic query algorithms. Running times in milliseconds.

Dataset	Synthetic						ChefMoz	GovTrack
#triples x 1000	10	50	250	1,250	6,250	10,000	549	12,340
Running time [ms]								
Consistency checking	21	235	1034	3461	10471	14381	1871	17541
Naive atomic query	12	276	4872	9761	27813	42315	10761	48913
atomicAnswerP	5	24	145	341	509	603	199	681
atomicAnswerS	11	185	451	491	1143	1576	475	1908
atomicAnswerV	10	176	481	567	1024	1456	510	1761
atomicAnswerA	2	10	98	131	154	236	104	341

In our experimental evaluation, we were particularly interested in studying the following:

- (1) The consistency checking running time on the GovTrack and ChefMoz datasets and its variation with the size of the synthetic dataset.
- (2) A comparison of the query answer time for all types of atomic queries and its variation with the size of the aRDF theory. A comparison with the naive query answer algorithm.
- (3) The comparison between the running times of *conjunctAnswer_GraphMatching* and *conjunctAnswer_Ordering* and their variations with the size of the query and size of the theory. A comparison with the naive conjunctive query algorithm.
- (4) An evaluation of the gain or loss in terms of running time of the view maintenance versus re-running the entire query.



(a) Running time of *consistencyCheck*, *naiveSimpleAnswer*, *answerA*, *answerP*, *answerV* and *answerS*

(b) Running time of *answerA*, *answerP*, *answerV* and *answerS*

Fig. 15. Consistency checking and atomic query answers

As a first step, we measured the running time of the consistency checking, naive atomic answer and the four atomic answer algorithms on all three datasets. For the query answer algorithms, the results are averaged over 50 separate queries over each dataset. A summary of the results is shown in Table I and a graphical depiction of the variation of the running time with the size of the synthetic dataset is shown in Figure 15(a) and (b) (in quasi-logscale). We point out that the most

Table II. Summary of conjunctive answer algorithms. Running times in milliseconds.

Dataset #triples x 1000	Query size	Synthetic						ChefMoz 549	GovTrack 12,340
		10	50	250	1,250	6,250	10,000		
Conjunct Graph Matching	5	45	214	876	2879	8917	11456	1243	15671
	10	79	394	1646	5243	15305	20452	2153	27980
	20	141	707	3078	9743	27319	38546	4064	47609
	30	243	1255	5443	17229	47106	70405	7255	87049
	40	454	2136	9721	32152	84949	123412	13240	157025
	50	830	3746	17033	56710	145076	227583	22705	294048
Conjunct Ordering	5	52	243	949	3141	10342	11606	1363	15947
	10	93	417	1709	5277	16552	21057	2581	29688
	20	108	504	2114	6407	27240	36155	2784	43286
	30	183	849	4038	17081	45968	46607	6613	79422
	40	369	1859	7831	30815	55734	90902	8897	147702
	50	721	2843	14739	48437	134872	208091	22403	204795

time consuming operation is by far the fixpoint computation – as seen from the data for the naive query answer algorithm. Given the logscale plot, the data suggests a nearly double exponential rise for the naive query algorithms. We can see that the consistency checking algorithm is much faster, primarily due to the fact that it avoids the fixpoint computation. Finally, from the atomic query answer algorithms, *atomicAnswerS* and *atomicAnswerV* take the longest. The motivation follows from the formal description of the algorithms: *atomicAnswerS* and *atomicAnswerV* search for p_q – paths originating at a known node r or ending in a known node v . On the other hand, for *atomicAnswerP* and *atomicAnswerA*, both r and v are known, which narrows the search space considerably.

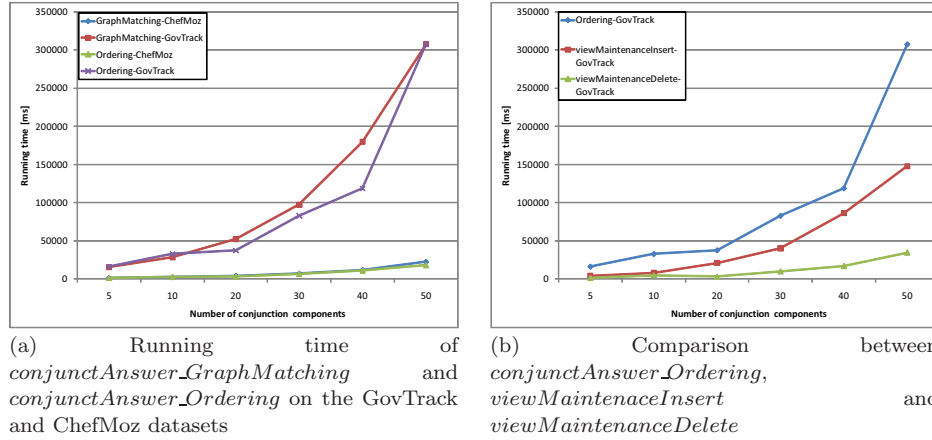


Fig. 16. Conjunctive queries and view maintenance

In the next step, we looked at *conjunctAnswer_GraphMatching*, *conjunctAnswer_Ordering* and *naiveConjunctAnswer*. We varied the size of the conjunctive queries from 5 to 50 component queries, while maintaining the number of variables in the graph patterns at 25%. This was done to ensure that the selectivity of the graph patterns remains stable (approximately 7%, with a standard deviation of .12%). The experimental results are summarized in Table II. The *naiveConjunctAnswer* (not

Table III. Summary of view maintenance algorithms. Running time in milliseconds.

Dataset #triples x 1000	Query size	Synthetic						ChefMoz 495	GovTrack 111,060
		9	45	225	1,125	5,625	9,000		
View maintenance Insert	5	22	38	414	1271	3812	2627	199	4137
	10	34	189	572	2017	5899	4574	734	7704
	20	62	294	1211	2184	5791	7700	1256	20745
	30	48	360	1892	3387	25995	14062	3253	40023
	40	105	351	4126	7562	23077	43606	3600	86134
	50	258	1476	6869	10362	59578	69393	5205	147840
View maintenance delete	5	4	6	110	253	780	935	71	1054
	10	8	34	389	363	1009	1101	188	4467
	20	6	107	168	690	2394	2752	206	2706
	30	25	88	316	2498	3046	6187	671	9600
	40	33	251	1973	4378	7590	26475	883	16700
	50	94	262	445	6929	21449	21338	2179	34395

shown in the table) ran out of memory at 1,250K triples for queries with 5,10 and 20 components, at 250K for queries with 30 and 40 components and at 50K triples for queries with 50 components. The running time for *naiveConjunctAnswer* algorithm was overwhelmingly larger than the other two algorithms (for instance, taking 7890 ms at 250K triples with 5 components compared to 876 and 949 ms for the graph matching and ordering algorithms respectively). In Figure 16(a) we can also see that the ordering algorithm does slightly better than the graph matching variant in terms of running time. We also notice that both algorithms have an average-case complexity much lower than the worst case complexity (which was factorial in the size of the data for the graph matching variant).

Finally, we looked at the view maintenance algorithms. We considered the same set of conjunctive queries as in the previous step and select uniformly at random 10% of the triples in each dataset to insert and respectively delete. We measured the running time of the view maintenance algorithms and give a summary of the results in Table III. Note that the size of dataset is the size before any insert operation and respectively after all the deletion operations. The cells in the table represent the average running time of the corresponding view maintenance algorithm after each insertion or deletion. The data clearly indicates that maintaining a conjunctive query can be done an order much faster than re-running the entire query. We also noticed that consistently, view maintenance when deleting a triple is much faster than view maintenance when inserting the same triple. This trend is also clear in Figure 16(b) for average insertion and deletion maintenance times. The fact that deletion maintenance is much faster than insertion maintenance can be explained by the difference in the number of decision points in the two algorithms: *viewMaintenanceDelete* has a lot fewer cases to consider than *viewMaintenanceInsert*.

8. RELATED WORK

There has been considerable work on extending RDF with new features such as time intervals (statements saying something is true at all time points in an interval [Gutiérrez et al. 2005]), uncertainty [Dubois et al. 2005; Straccia 2005] (though these are just one page position papers) and provenance — [Carroll et al. 2005] describes a model for representing named RDF graphs, thus allowing statements

about RDF graphs to be represented in RDF. [Gergatsoulis and Lilis 2005] defines a model for representing multi-dimensional RDF, where information can be context dependent; for instance the title of a book may be represented in different languages. Our approach differs from all of the above: (i) we define a general framework for extending the RDF data model with annotations from an arbitrary partially ordered set; (ii) we give a single unified algorithm to assess consistency of such theories, and (iii) we give efficient algorithms for querying annotated RDF theories and (iv) we provide view maintenance algorithms in such cases. None of the above papers do (iii) and (iv) even for the specific type of RDF extension (e.g. temporal, provenance, that they provide). To the best of our knowledge, this is the first paper that has attempted to provide a single framework - where by swapping a new partial order (with bottom) for another - we can get different types of reasoning capabilities in RDF. We have shown that annotated RDF is capable of supporting diverse forms of reasoning as well as combinations of reasoning (e.g. via `fuztime`), has a rich declarative semantics, and provides an efficient computational engine for application building.

aRDF is built on top of annotated logic that has been extensively studied [Kifer and Subrahmanian 1992; Krishnaprasad and Kifer 1993; S. Krajci and Vojtas 2004; Lu et al. 2002]. This is the first time that annotated logic is being used to extend RDF. There are several differences from past work in annotated logic. (i) The results for aRDF apply to partially ordered sets - work on annotated logic primarily used complete lattices and semi-lattices. (ii) Though there is extensive work on query processing for annotated logic, we are not aware of any algorithms that focus on the class of annotated logic theories that resemble aRDF theories - the algorithms in this paper apply to this smaller class and are much more efficient than, say, the algorithms in [Kifer and Subrahmanian 1992]. (iii) There is no work we are aware of on view maintenance in annotated logic. (iv) We are aware of no experimental results in annotated logic that rival the sizes (over 12 million triples) that we have exhibited in this paper.

Work on view maintenance in RDF is relatively minuscule. Volz et. al. [R. Volz and Studer 2002] were the first to introduce views into RDF. They required that the results of queries contain class instances and that the result itself has the pattern of an RDF statement. We do not have these requirements. Magnaraki et. al. [Magkanaraki et al. 2003] proposed RVL - a language for RDF views. However, they do not address the view maintenance problem. Hung et. al. [E. Hung and Subrahmanian 2005] present a mechanism to handle aggregate queries and update aggregate views over RDF databases - we do not consider aggregates in this paper. None of these worthwhile papers address the complex view maintenance problems that arise when RDF is extended through a unified extension method that accommodates temporal, fuzzy-temporal, provenance, and other kinds of annotations. By using annotated RDF, we were able to present view maintenance algorithms that directly applied to such extensions of RDF rather than having to invent such algorithms separately for each of them.

9. CONCLUSIONS

There are now many rapidly emerging extensions of RDF. Examples include fuzzy extensions to RDF [Mazzieri and Dragoni 2005], temporal extensions [Gutiérrez et al. 2005], possibilistic extensions [Dubois et al. 2005; Straccia 2005], and extensions to handle provenance [Carroll et al. 2005]. In this paper, we have provided a single theoretical framework called *annotated RDF*, or *aRDF* to capture such extensions. *aRDF* provides a semantics for RDF augmented with a partially ordered set and show that this semantics can capture the aforementioned types of reasoning. In particular, new partially ordered sets can be plugged into *aRDF* depending upon the application and there will be an immediate set of results that apply to that domain.

More importantly, we have provided algorithms for query processing and view maintenance for any *aRDF* theory. In particular, this yields new results even for temporal RDF and RDF with provenance for which no view maintenance algorithms exists. We have shown the correctness and complexity of our algorithms for arbitrary *aRDF* settings. More importantly, we have shown that our algorithms can scale well, handling queries to real world, very large data sets containing over 12 million triples.

REFERENCES

- CALI, A. AND LUKASIEWICZ, T. Tightly integrated probabilistic description logic programs for the semantic web. In *ICLP 2007*. 428–429.
- CARROLL, J. J., BIZER, C., HAYES, P., AND STICKLER, P. 2005. Named graphs, provenance and trust. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*. ACM Press, New York, NY, USA, 613–622.
- CORDELLA, L. P., FOGGIA, P., SANSONE, C., AND VENTO, M. 2004. A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs. *IEEE Trans. PAMI* 26, 10, 1367–1372.
- DUBOIS, D., MENGIN, AND PRADE, H. 2005. Possibilistic uncertainty and fuzzy features in description logic: a preliminary discussion. In *Proc. Workshop on Fuzzy Logic and the Semantic Web (ed. E. Sanchez)*. 5–7.
- E. HUNG, Y. D. AND SUBRAHMANIAN, V. S. 2005. Rdf aggregate queries and views. In *Proc. IEEE Intl. Conf. on Data Engineering*. 717–728.
- FITTING, M. 1991. Bilattices and the semantics of logic programming. *J. Log. Program.* 11, 2, 91–116.
- FLOYD, R. 1962. Algorithm 97: Shortest path. *Communications of the ACM* 5, 6, 345.
- G. ANTONIOU, F. V. H. 2004. Web ontology language:owl. In *Handbook of Ontologies*. 67–92.
- GERGATSOULIS, M. AND LILIS, P. 2005. Multidimensional rdf. In *Proc. 2005 Intl. Conf. on Ontologies, Databases, and Semantics (ODBASE)*. Vol. 3761. Springer, 1188–1205.
- GUPTA, A. AND MUMICK, I. 1999. *Materialized Views: Techniques, Implementations, and Applications*. MIT Press.
- GUTIÉRREZ, C., HURTADO, C. A., AND VAISMAN, A. A. 2005. Temporal rdf. In *ESWC*. 93–107.
- HALASCHEK-WIENER, C., GOLBECK, J., SCHAIN, A., GROVE, M., PARSIA, B., AND HENDLER, J. 2006. Annotation and provenance tracking in semantic web photo libraries. In *IPAW 2006*. 82–89.
- HLAOUI, A. AND WANG, S. 2002. A new algorithm for inexact graph matching. In *ICPR (4)*. 180–183.
- KAHAN, J. AND KOIVUNEN, M.-R. 2001. Annotea: an open rdf infrastructure for shared web annotations. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*. ACM Press, New York, NY, USA, 623–632.

- KARVOUNARAKIS, G., ALEXAKI, S., CHRISTOPHIDES, V., PLEXOUSAKIS, D., AND SCHOLL, M. 2002. Rql: a declarative query language for rdf. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*. ACM Press, New York, NY, USA, 592–603.
- KIFER, M. AND SUBRAHMANIAN, V. S. 1992. Theory of generalized annotated logic programming and its applications. *J. Log. Program.* 12, 4, 335–367.
- KRISHNAPRASAD, T. AND KIFER, M. 1993. A theory of nonmonotonic inheritance based on annotated logic. *Artificial Intelligence* 60, 1, 23–50.
- LEACH, S. M. AND LU, J. J. 1996. Query processing in annotated logic programming: Theory and implementation. *J. Intell. Inf. Syst.* 6, 1, 33–58.
- LU, J. J., MURRAY, N. V., RADJAVI, H., ROSENTHAL, E., AND ROSENTHAL, P. 2002. Inference for annotated logics over distributive lattices. In *ISMIS*. 285–293.
- LUKASIEWICZ, T. AND STRACCIA, U. Tightly integrated fuzzy description logic programs under the answer set semantics for the semantic web. In *RR 2007*. 289–298.
- MADUKO, A., ANYANWU, K., SHETH, A., AND SCHLIEKELMAN, P. 2007. Estimating the cardinality of rdf graph patterns. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*. ACM Press, New York, NY, USA, 1233–1234.
- MAGKANARAKI, A., TANNEN, V., CHRISTOPHIDES, V., PLEXOUSAKIS, D., SCHOLL, M., AND TOLLE, R. 2003. Viewing the semantic web with rvl lenses. In *Proc. Intl. Semantic Web Conference*.
- MARTELLI, A. AND MONTANARI, U. 1982. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems* 4, 2, 258–282.
- MAZZIERI, M. AND DRAGONI, A. F. 2005. A fuzzy semantics for semantic web languages. In *ISWC-URSW*. 12–22.
- R. VOLZ, D. O. AND STUDER, R. 2002. Towards views in the semantic web. In *Proc. 2nd Intl. Workshop on Databases, Documents, and Information Fusion*.
- S. KRAJCI, R. L. AND VOJTAS, P. 2004. A comparison of fuzzy and annotated logic programming. *Fuzzy Sets and Systems* 144, 1, 173–192.
- STRACCIA, U. 2005. Towards a fuzzy description logic for the semantic web. In *Proc. Workshop on Fuzzy Logic and the Semantic Web (ed. E. Sanchez)*. 3–3.
- UDREA, O., RECUPERO, D. R., AND SUBRAHMANIAN, V. S. 2006. Annotated rdf. In *ESWC*. 487–501.