Query Relaxation in RDF

Carlos A. Hurtado[1,⋆], Alexandra Poulovassilis[2], and Peter T. Wood[2]

[1] Universidad de Chile
churtado@dcc.uchile.cl
[2] Birkbeck, University of London
{ap,ptw}@dcs.bbk.ac.uk

Abstract. We explore flexible querying of RDF data, with the aim of making it possible to return data satisfying query conditions with varying degrees of exactness, and also to rank the results of a query depending on how "closely" they satisfy the query conditions. We make queries more flexible by logical relaxation of their conditions based on RDFS entailment and RDFS ontologies. We develop a notion of ranking of query answers, and present a query processing algorithm for incrementally computing the relaxed answer of a query. Our approach has application in scenarios where there is a lack of understanding of the ontology underlying the data, or where the data objects have heterogeneous sets of properties or irregular structures.

1 Introduction

The conjunctive fragment of most RDF query languages (e.g., see [10,11]) consists of queries of the form $H \leftarrow B$, where the body of the query B is a graph pattern, that is, an RDF graph over IRIs, literals, blanks, and variables. The head of the query H is either a graph pattern or a tuple variable (list of variables). The semantics of these queries is simple. It is based on finding matchings from the body of the query to the data and then applying the matchings to the head of the query to obtain the answers.

Recently, the W3C RDF data access group has emphasized the importance of enhancing RDF query languages to meet the requirements of contexts where RDF can be used to solve real problems. In particular, it has been stated that in RDF querying "it must be possible to express a query that does not fail when some specified part of the query fails to match" [5]. This requirement has motivated the `OPTIONAL` clause, presented in the emerging SPARQL W3C proposal for querying RDF [17] and previously introduced in SeRQL [3]. The `OPTIONAL` clause allows the query to find matchings that fail to match some conditions in the body. In contrast to other approaches to flexible querying (e.g., [1,14]), the `OPTIONAL` construct incorporates flexibility from a "logical" standpoint, via relaxation of the query's conditions. This idea, however, is exploited only to a limited extent, since the conditions of a query could be relaxed in ways

⋆ Carlos A. Hurtado was supported by Millennium Nucleus, Center for Web Research (P04-067-F), Mideplan, and by project FONDECYT 1030810, Chile.

S. Spaccapietra (Ed.): Journal on Data Semantics X, LNCS 4900, pp. 31–61, 2008.

other than simply dropping optional triple patterns, for example by replacing constants with variables or by using the class and property hierarchies in an ontology associated with the data (such as that shown in Figure 1).

In this paper, we propose the introduction of a `RELAX` clause as a generalization of the `OPTIONAL` clause for the conjunctive fragment of SPARQL. The idea is to make queries more flexible by a logical relaxation of some of the conditions that are enclosed by one or more `RELAX` clauses inside the body of the query. These conditions are successively turned more general so that the query is transformed and processed to successively return more general answers. We define the notion of "being more general" (or "being more relaxed") using RDFS entailment and RDFS ontologies.

1.1 RDFS Ontologies

It is common that users interact with RDF applications in the context of an ontology. As an example, OWL-QL [8] allows users to include ontologies as premises in queries, and SPARQL provides a similar facility by allowing reference to several RDF datasets [17] in a query. As we will show later, ontologies provide an important source of knowledge to support query relaxation.

Before addressing the central ideas of our approach, we give a brief description of the type of ontologies we will consider. We assume that a query is interpreted in the context of a single ontology, which is modeled as an RDF graph with interpreted RDFS vocabulary. The RDFS vocabulary defines classes and properties that may be used for describing groups of related resources and relationships between resources. In this paper we use a fragment of the RDFS vocabulary, which comprises (in brackets is the shorter name we will use) rdfs:range [`range`], rdfs:domain [`dom`], rdf:type [`type`], rdfs:subClassOf [`sc`] and rdfs:subPropertyOf [`sp`][1].

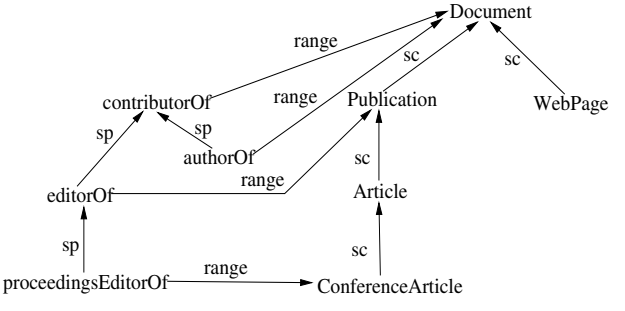

Fig. 1. An RDFS ontology modeling documents and people who contribute to them

[1] We omit in this paper vocabulary used to refer to basic classes in RDF/S such as rdf:Property, rdfs:Class, rdfs:Resource, rdfs:Literal, rdfs:XMLLiteral, rdfs:Datatype, among others. We also omit vocabulary for lists, collections, and variations on these, as well as vocabulary used to place comments in RDF/S data.

As an example, the ontology of Figure 1 is used to model documents along with properties that model different ways people contribute to them (e.g., as authors, editors, or being the editor of the proceedings where an article is published).

1.2 The RELAX Clause

We now explain the RELAX as an extension of the OPTIONAL clause. As an example, consider the following SPARQL-like query Q[2]:

$$?Z, ?Y \leftarrow \{(?X, name, ?Z), \texttt{OPTIONAL}\{(?X, proceedingsEditorOf, ?Y)\}\}.$$

The body of this query is a graph pattern comprising two triple patterns. This query returns names of people along with the IRIs of conference articles whose proceedings they have edited. Because the second triple pattern in the body of the query is within the scope of an OPTIONAL clause, the query also returns names of people for which the second pattern fails to match the data (i.e., people who have not edited proceedings).

Now, instead of dropping the triple pattern $(?X, proceedingsEditorOf, ?Y)$ we may relax this triple pattern by using the ontology of Figure 1. As an example, though the user may want to retrieve editors of proceedings at first, she/he might also be interested in knowing about people who have contributed to publications in other roles, along with the publications themselves. Now after returning editors of conference proceedings, the user could replace the triple pattern $(?X, proceedingsEditorOf, ?Y)$ with $(?X, editorOf, ?Y)$, yielding a new, relaxed query that returns editors of publications along with their publications. Subsequently, this triple pattern can be rewritten to the triple pattern $(?X, contributorOf, ?Y)$ to obtain more general answers.

In order to save the user the effort of inspecting the ontology and rewriting the query to return more relaxed answers for the same original query, the system could perform this process automatically. This is achieved by the following query which replaces OPTIONAL with RELAX in Q:

$$?Z, ?Y \leftarrow \{(?X, name, ?Z), \texttt{RELAX}\{(?X, proceedingsEditorOf, ?Y)\}\}.$$

The idea of making queries more flexible by the logical relaxation of their conditions is not new in database research. Gaasterland et al. [9] proposed a mechanism to achieve this goal in the context of deductive databases and logic programming, and called the technique *query relaxation*.

1.3 Notion of Query Relaxation for RDF

We study the query relaxation problem in the setting of the RDF/S data model and RDF query languages and show that query relaxation can be naturally formalized using RDFS entailment. We use an operational semantics for the notion of RDFS entailment, denoted $\models$, characterized by the derivation rules

[2] SPARQL has SQL-like syntax; for brevity, in this paper we express queries as rules.

Group A (Subproperty) (1) $\frac{(a,\texttt{sp},b)\ \ (b,\texttt{sp},c)}{(a,\texttt{sp},c)}$ (2) $\frac{(a,\texttt{sp},b)\ \ (x,a,y)}{(x,b,y)}$

Group B (Subclass) (3) $\frac{(a,\texttt{sc},b)\ \ (b,\texttt{sc},c)}{(a,\texttt{sc},c)}$ (4) $\frac{(a,\texttt{sc},b)\ \ (x,\texttt{type},a)}{(x,\texttt{type},b)}$

Group C (Typing) (5) $\frac{(a,\texttt{dom},c)\ \ (x,a,y)}{(x,\texttt{type},c)}$ (6) $\frac{(a,\texttt{range},d)\ \ (x,a,y)}{(y,\texttt{type},d)}$

Fig. 2. RDFS Inference Rules

given in Figure 2 (for details, see [10,12]). The rules describe the semantics of the RDFS vocabulary we use in this paper (i.e., `sp`, `sc`, `type`, `dom`, and `range`)[3].

Intuitively, as RDFS entailment is characterized by the rules of Figure 2, a relaxed triple pattern t' can be obtained from triple t by applying the derivation rules to t and triples from the ontology. As an example, the triple pattern $(?X, \mathit{proceedingsEditorOf}, ?Y)$ can be relaxed to $(?X, \mathit{editorOf}, ?Y)$, by applying rule 2 to the former and the triple $(\mathit{proceedingsEditorOf}, \texttt{sp}, \mathit{editorOf})$ in the ontology of Figure 1. The different relaxed versions of an original query are obtained by combining relaxations of triple patterns that appear inside a `RELAX` clause.

The notion of query relaxation we propose naturally subsumes two broad classes of relaxations. The first class of relaxations includes relaxations entailed using information from the ontology and are captured by the rules of Figure 2; these include relaxing type conditions, relaxing properties using domain or range restrictions and others. The second class of relaxation consists of relaxations that can be entailed without an ontology, which include dropping triple patterns, replacing constants with variables, and breaking join dependencies.

1.4 Summary of Contributions

In this paper, we develop a framework for query relaxation for RDF. We introduce a notion of query relaxation based on RDFS entailment, which naturally incorporates RDFS ontologies and captures necessary information for relaxation such as the class and property hierarchies.

By formalizing query relaxation in terms of entailment, we obtain a semantic notion which is by no means limited to RDFS and could also be extended to more expressive settings such as OWL entailment and OWL ontologies, to capture further relaxations. Our framework generalizes, for the conjunctive fragment of SPARQL, the idea of dropping query conditions provided by the `OPTIONAL` construct.

An essential aspect of our proposal, which sets it apart from previous work on query relaxation, is to rank the results of a query based on how "closely" they

[3] We omit, for now, the RDFS rule that essentially states that blank nodes (or variables) behave like existentially quantified variables, and allows constants to be replaced with blanks or blanks with other blanks using the notion of a *map*. This notion and its use will be covered in Section 6.

satisfy the query. We present a notion of ranking based on a structure called the *relaxation graph*, in which relaxed versions of the original query are ordered from less to more general from a logical standpoint. Since the relaxation graph is based on logical subsumption, ranking does not depend on any syntactic condition on the knowledge used for relaxation (such as rule ordering in logic-programming approaches [9]). Finally, we give a query processing algorithm to compute the relaxed answer of a query, and examine its correctness and complexity.

This paper extends our earlier paper [13] in a number of ways. We have substantially developed, revised and improved the material presented there. We also make the following new contributions here: we provide proofs for all the results sketched in [13]; Section 4 includes substantial new contributions relating to the relationship between relaxations and derivations using new RDFS rules; we provide a new algorithm for computing relaxations based on the notion of the "extended reduction" of the ontology used for relaxation; and Section 5, on computing relaxed query answers, has been extended with two examples that illustrates our query processing algorithm.

1.5 Outline

The rest of the paper is organized as follows. Section 2 introduces preliminary notation. We then present our framework in a stepwise manner. Firstly, in sections 3, 4, and 5, we formalize and study relaxations that do not replace terms of the original triple pattern with variables and are captured by the rules of Figure 2; they include relaxing type conditions, relaxing properties using domain or range restrictions and others. In particular, in Section 3 we formalize the semantics of query relaxation for the aforementioned types of relaxation. Then, Section 4 discusses the problem of computing relaxations of a triple pattern and Section 5 studies query processing. In Section 6, we extend the relaxation framework to consider relaxations that replace terms of the original triple pattern with variables (e.g., replacing a literal or IRI with a variable or a variable with another variable). In Section 7 we review related work in comparison to our own work, and we give our concluding remarks in Section 8. Finally, in the appendix we present the proofs omitted in the main body of the paper.

2 Preliminary Definitions

In this section we present the basic notation and definitions that will be used subsequently in this paper. Some of these were introduced in [2,10,12,15].

2.1 RDF Graphs and RDFS Ontologies

In this paper we work with RDF graphs which may mention the RDFS vocabulary. We assume there are infinite sets I (IRIs), B (blank nodes), and L (RDF literals). The elements in $I \cup B \cup L$ are called RDF *terms*. A triple $(v_1, v_2, v_3) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an *RDF triple*. In such a triple,

v_1 is called the *subject*, v_2 the *predicate* and v_3 the *object*. An *RDF graph* (just graph from now on) is a set of RDF triples.

We consider ontologies that use RDFS vocabulary, which we will refer to as RDFS ontologies. We assume that predicates of triples in O should be in the set $\{\texttt{type}, \texttt{dom}, \texttt{range}, \texttt{sp}, \texttt{sc}\}$. Intuitively, this means that the ontology does not interfere with the semantics of the RDFS vocabulary.

We say that an ontology is acyclic if the subgraphs defined by `sc` and `sp` are acyclic. Acyclicity is considered good practice in modeling ontologies.

We write that $G_1 \models_{\texttt{rule}} G_2$ if G_2 can be derived from G_1 by iteratively applying the rules of Figure 2. In this paper, we also use a notion of closure of an RDF graph G [12], denoted $\text{cl}(G)$, which is the closure of G under the rules. By a result from [12], RDFS entailment (for the fragment of RDFS we use in this paper) can be characterized as follows: $G_1 \models_{\texttt{RDFS}} G_2$ if and only if $G_2 \subseteq \text{cl}(G_1)$.

2.2 Conjunctive Queries for RDF

Consider a set of variables V disjoint from the sets I, B, and L. A *triple pattern* is a triple $(v_1, v_2, v_3) \in (I \cup V) \times (I \cup V) \times (I \cup V \cup L)$. A *graph pattern* is a set of triple patterns. Given a graph pattern P, we denote by $\texttt{var}(P)$ the variables mentioned in P. In our examples, variables are indicated by a leading question mark, while literals are enclosed in quotes.

A *conjunctive query* Q is an expression $T \leftarrow B$, where B is a graph pattern, and $T = \langle T_1, \ldots, T_n \rangle$ is a list of variables which belongs to $\texttt{var}(B)$. (The framework formalized in this paper can be easily extended to queries with graph patterns as query heads.) We denote T by $\texttt{Head}(Q)$, and B by $\texttt{Body}(Q)$.

We next define the answer of a conjunctive query. In order to do this, we take into account that a query Q may be formulated over an RDFS ontology O, which means that Q may mention vocabulary from O and its answer is obtained from the RDF graph being queried and O. We define a *matching* to be a function from variables in $\texttt{Body}(Q)$ to blanks, IRIs and literals. Given a matching Θ, we denote by $\Theta(\texttt{Body}(Q))$ the graph resulting from $\texttt{Body}(Q)$ by replacing each variable X by $\Theta(X)$. Given an RDF graph G, the *answer* of Q is the set of tuples, denoted $\texttt{ans}(Q, O, G)$, defined as follows: for each matching Θ such that $\Theta(\texttt{Body}(Q)) \subseteq \text{cl}(O \cup G)$, return $\Theta(\texttt{Head}(Q))$. When O is clear from the context, we omit it, and write $\texttt{ans}(Q, G)$ instead of $\texttt{ans}(Q, O, G)$.

3 Formalizing Query Relaxation

We will present a relaxed semantics for queries in a stepwise manner. In Section 3.1, we present the notion of relaxation of triple patterns, and in Section 3.2 we introduce the notion of the relaxation graph of a triple pattern. This is used in Section 3.3 to define the relaxation graph of a query. In Section 3.4, we explain different types of relaxations subsumed by our framework. The relaxation graph is the basis for the notion of the relaxed answer and ranking of a query we propose in Section 3.5.

3.1 Triple Pattern Relaxation

In this section, we define the relaxation relation between triple patterns. Intuitively, a triple pattern relaxes to another triple pattern if the latter can be logically derived from the former and a given ontology. Relaxation will be defined in the context of an ontology that will be denoted by O.

Definition 1 (Triple Pattern Relaxation). *Let t_1, t_2 be triple patterns such that $t_1, t_2 \notin \mathrm{cl}(O)$, and $\mathtt{var}(t_2) = \mathtt{var}(t_1)$. We say that t_1 relaxes to t_2 (or t_2 is a relaxation of t_1), denoted $t_1 \leq t_2$, if $(\{t_1\} \cup O) \models_{\mathtt{rule}} t_2$.*

As stated before, in this section we consider relaxations that maintain the set of variables in the original triple pattern. This is formalized in the previous definition by requiring that $\mathtt{var}(t_2) = \mathtt{var}(t_1)$. In addition, we require that $t_1, t_2 \notin \mathrm{cl}(O)$ in order to avoid relaxing to triple patterns that will be trivially true for any RDF graph being queried.

As an example, let O be the ontology of Figure 1. Then, we have that

$$(?X, \mathtt{type}, \mathit{ConferenceArticle}) \leq (?X, \mathtt{type}, \mathit{Article})$$

and

$$(\mathit{JohnRobert}, \mathit{ContributorOf}, ?X) \leq (?X, \mathtt{type}, \mathit{Document})$$

among other relaxations. It is not the case, however, that

$$(?X, \mathit{ContributorOf}, ?Y) \leq (?Y, \mathtt{type}, \mathit{Document})$$

since the sets of variables in the two triple patterns are different.

The following proposition shows that triple pattern relaxation can be characterized in terms of the RDFS closure.

Proposition 1. *Let $\leq$ be defined using an ontology O, and t_1, t_2 be triple patterns such that $t_1, t_2 \notin \mathrm{cl}(O)$ and $\mathtt{var}(t_1) = \mathtt{var}(t_2)$. Then $t_1 \leq t_2$ if and only if $t_2 \in \mathrm{cl}(O \cup \{t_1\})$.*

It is desirable that the relaxation relation should be a partial order. The following proposition shows the conditions under which this happens.

Proposition 2. *Let $\leq$ be defined using an ontology O. Then $\leq$ is a partial order if and only if O is acyclic.*

In what follows we assume that O is acyclic. Therefore, from now on we assume that the relaxation relation is a partial order.

The *direct relaxation relation*, denoted by $\prec$, is the reflexive and transitive reduction of $\leq$. The *direct relaxations* of a triple pattern t (i.e., triples t' such that $t \prec t'$) are important in our framework, since they are the result of the smallest steps of relaxation. The *indirect relaxations* of a triple pattern t are the triples t' such that $t \leq t'$ and $t \not\prec t'$.

3.2 Relaxation Graph of a Triple Pattern

We are interested in relaxing each of the triple patterns that occurs inside the `RELAX` clause of a query, so we next adapt the relaxation relation to use relaxation "above" a given triple pattern. This yields the notion of relaxation graph of a triple pattern.

Definition 2 (Relaxation Graph of a Triple Pattern). *The relaxation relation (resp., direct relaxation relation) "above" a triple pattern t, denoted by $\leq_t$ (resp., $\prec_t$), is $\leq$ (resp., $\prec$) restricted to triple patterns t' such that $t \leq t'$. The relaxation graph of a triple pattern t is the directed acyclic graph induced by $\prec_t$.*

As an example, Figure 3 shows the relaxation graph of $(JohnRobert, editorOf, ?X)$, assuming that O is the ontology of Figure 1.

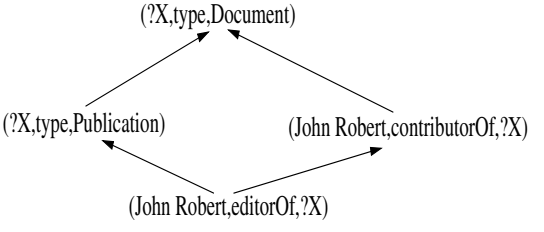

Fig. 3. Relaxation graph of the triple pattern $(JohnRobert, editorOf, ?X)$

3.3 Relaxation Graph of a Query

We now generalize triple pattern relaxation to query relaxation using the notion of the direct product of partial orders. The *direct product* of two partial order relations α_1, α_2, denoted $\alpha_1 \otimes \alpha_2$, is another partial order α such that $(a, b)\ \alpha\ (c, d)$ if and only if $a\ \alpha_1\ c$ and $b\ \alpha_2\ d$. The generalization of this definition to more than two relations is straightforward.

Definition 3 (Relaxation Graph of a Query). *Given a query Q, let* $\texttt{Body}(Q) = \{t_1, \ldots, t_n\}$. *For any triple t_i not inside a* `RELAX` *clause, we overload the notation $\leq_{t_i}$ and assume that t_i relaxes only to t_i. Then, the relaxation relation "above" Q, denoted by $\leq_Q$, is defined as $\leq_{t_1} \otimes \leq_{t_2} \ldots \otimes \leq_{t_n}$. Direct relaxation, denoted $\prec_Q$, is the reflexive and transitive reduction of $\leq_Q$. The relaxation graph of Q is the directed acyclic graph induced by $\prec_Q$.*

It is important to remark that a node $(t'_1, \ldots, t'_n)$ in the relaxation graph of Q denotes the conjunctive query $\texttt{Head}(Q) \leftarrow t'_1, \ldots, t'_n$.

As an example, consider the following query:

$$?X \leftarrow \{\texttt{RELAX}\{(?X, \texttt{type}, Publication)\}, \texttt{RELAX}\{(JohnRobert, editorOf, ?X)\}.$$

Figure 4 (A) shows the relaxation graph of each of the triple patterns of the query (for the sake of space, we consider in this example only a single edge of the relaxation graph of $(JohnRobert, editorOf, ?X)$). Figure 4 (B) shows the direct product of the graphs of Figure 4 (A), which is a simplified version of the relaxation graph of the query.

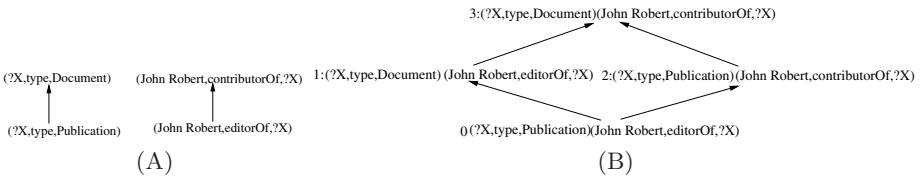

Fig. 4. (A) The relaxation graph of $(?X, \texttt{type}, \mathit{Publication})$ and a simplified version of the relaxation graph of $(\mathit{JohnRobert}, \mathit{editorOf}, ?X)$. (B) The direct product of the graphs given in (A). Nodes are enumerated from 0 (base query) to 3 (top query).

3.4 Types of Relaxation

The notion of relaxation that we have presented in this section encompasses the following types of relaxation (the examples given use the ontology of Figure 1):

1. Type relaxation: replacing a triple pattern $(a, \texttt{type}, b)$ with $(a, \texttt{type}, c)$, where $(b, \texttt{sc}, c) \in \mathrm{cl}(O)$. For example, the triple pattern $(?X, \texttt{type}, \mathit{ConferenceArticle})$ can be relaxed to $(?X, \texttt{type}, \mathit{Article})$ and then to $(?X, \texttt{type}, \mathit{Publication})$.
2. Predicate relaxation: replacing a triple pattern (a, p, b) with (a, q, c), where $(p, \texttt{sp}, q) \in \mathrm{cl}(O)$. For example, the triple pattern $(?X, \mathit{proceedingsEditorOf}, ?Y)$ can be relaxed to $(?X, \mathit{editorOf}, ?Y)$ and then to $(?X, \mathit{contributorOf}, ?Y)$.
3. Predicate to domain relaxation: replacing a triple pattern (a, p, b) with $(a, \texttt{type}, c)$, where $(p, \texttt{dom}, c) \in \mathrm{cl}(O)$. There are no domain declarations in Figure 1.
4. Predicate to range relaxation: replacing a triple pattern (a, p, b) with $(b, \texttt{type}, c)$, where $(p, \texttt{range}, c) \in \mathrm{cl}(O)$. For example, the triple pattern $(\mathit{JohnRobert}, \mathit{editorOf}, ?Y)$ can be relaxed to $(?Y, \texttt{type}, \mathit{Publication})$.
5. Additional relaxations induced by additional rules from Figure 2. Combinations of rules yield additional forms of relaxation. For example, the triple pattern $(\mathit{Article}, \texttt{sc}, ?Y)$ can be relaxed to $(\mathit{ConferenceArticle}, \texttt{sc}, ?Y)$.

3.5 Notion of Ranking

An algorithm used to process a query with a `RELAX` clause should return a list of tuples. A condition of consistency for the algorithm is that the tuples that are computed by more specific queries should appear before the ones that are computed by more general queries. If this happens, we say that the algorithm returns its answer in *ranked order*. In this section we formalize this idea.

In order to formalize this idea, we firstly define, for a query Q' in the relaxation graph of Q, the set of tuples returned by Q' and not returned by queries below Q'. We call such a set the *new answer* of Q'. Formally,

$$\texttt{newAnswer}(Q', G) := \texttt{ans}(Q', G) - (\textstyle\bigcup_{Q_i : Q_i \leq_Q Q', Q_i \neq Q'} \texttt{ans}(Q_i, G)).$$

Definition 4 (Ranking). *Consider an algorithm A that, given a query Q and an RDF graph, returns a list of tuples $L = a_1, \ldots, a_n$. We say that A returns its*

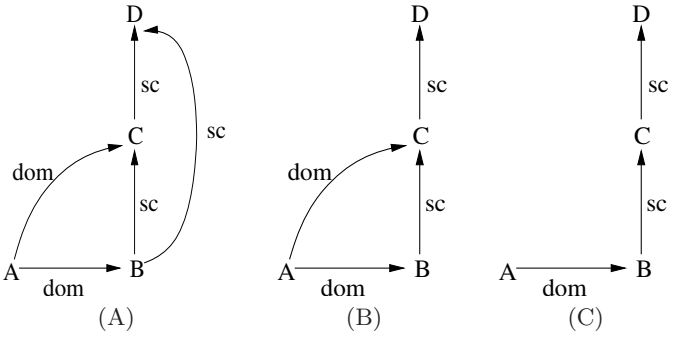

Fig. 5. (A) An ontology O. (B) The reduction $\texttt{red}(O)$ of O. (C) The extended reduction $\texttt{extRed}(O)$ of O.

tuples in ranked order if (i) the set $\{a_1, \ldots, a_n\}$ *is equal to* $\bigcup_{Q_i : Q \leq_Q Q_i} \texttt{ans}(Q_i, G)$*, and (ii) for all pairs of queries* Q_i, Q_j *such that* $Q_i \prec_Q Q_j$*, the new answers of* Q_i *appear earlier in* L *than the new answers of* Q_j*.*

4 Computing the Relaxation Graph

In this section, we study the problem of computing the relaxation graph of a triple pattern. In Section 4.1 we present a naive procedure to do so. Then, in Section 4.2, we show an efficient algorithm to perform this task. Finally, in Section 4.3, we study the size of the relaxation graph and the complexity of computing it.

4.1 Computing the Relaxation Graph of a Triple Pattern: Naive Algorithm

As Proposition 1 shows, it is possible to generate all the relaxations of a triple pattern t by computing $\text{cl}(O \cup \{t\})$ and $\text{cl}(O)$. However, recall that the edges of the relaxation graph are direct relaxations, and therefore the fundamental problem we need to solve is how to efficiently generate the direct relaxations of t. One may naively attempt to generate them by applying the derivation rules of Figure 2 over t and triples from the ontology $\text{cl}(O)$. We write $t, o \vdash t'$ if t' can be derived from t and $o \in \text{cl}(O)$ by the application of a single rule from Figure 2. We also write $t, o \vdash_i t'$ if rule i was the rule used in the derivation.

As an example, let O be the ontology given in Figure 5 (A). Notice that in this case $O = \text{cl}(O)$. Now, the following instantiation of rule 4

$$(B, \texttt{sc}, C), (?X, \texttt{type}, B) \vdash_4 (?X, \texttt{type}, C),$$

produces the direct relaxation $(?X, \texttt{type}, C)$ of $(?X, \texttt{type}, B)$. However, the following instantiation of rule 4

$$(B, \texttt{sc}, D), (?X, \texttt{type}, B) \vdash_4 (?X, \texttt{type}, D),$$

produces the indirect relaxation $(?X, \texttt{type}, D)$ of $(?X, \texttt{type}, B)$.

The example shows that the application of rules over the closure of the ontology and the triple t is not correct for computing direct relaxations of t. However, the next proposition shows that the procedure is complete for this purpose.

Proposition 3. *Let t_a, t_b be triple patterns not in* $\mathrm{cl}(O)$ *such that* $\texttt{var}(t_a) = \texttt{var}(t_b)$. *If $t_a \prec t_b$ then there exists a triple $o \in \mathrm{cl}(O)$ such that $t_a, o \vdash t_b$.*

As an aside, Proposition 3 along with the rules of Figure 2 allows us to classify the direct relaxations that can be obtained from a given triple pattern, which is done in Figure 6. We refer to triples having `type` as their predicate as `type` *triples*. We use a similar notation for triples with predicates `dom`, `range`, `sp`, and `sc`. A *plain triple* is a triple whose predicate term is not in the RDFS vocabulary. As an example, the figure shows that a direct relaxation of a plain triple is either a plain triple or a `type` triple. Notice that neither `dom` triples nor `range` triples can be relaxed.

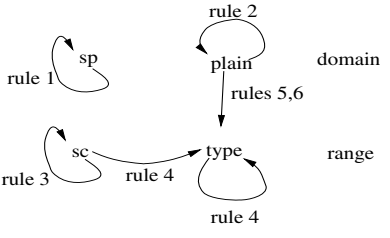

Fig. 6. Diagram of possible direct relaxations

In the remainder of the section, we present a naive algorithm to compute the relaxation graph of a triple pattern.

Firstly, let us introduce some notation to refer to the procedure just outlined that uses the RDFS derivation rules to produce relaxations of a triple pattern. We denote by $\texttt{applyRules}(t, O)$ all the triples generated by instantiating a rule of Figure 2 with t and a triple from O. Using O from Figure 5 (A) as an example, $\texttt{applyRules}((?X, \texttt{type}, B), O)$ generates the triple patterns $(?X, \texttt{type}, C)$ and $(?X, \texttt{type}, D)$. We will frequently abuse notation and consider $\texttt{applyRules}(t, O)$ simply as a set. Furthermore, since relaxations preserve variables of t, we assume we filter from $\texttt{applyRules}(t, O)$ triples t' such that $\texttt{var}(t') \neq \texttt{var}(t)$.

The naive algorithm works as follows. In a first step, it builds a graph that subsumes the relaxation graph of t. This graph may contain some indirect relaxations that need to be deleted in a second step. In the first step, it start by calling $\texttt{applyRules}(t, O)$, update the graph with new relaxations of t, and adds them to a list. It then removes a triple pattern t' from the list, update the graph with new relaxations $\texttt{applyRules}(t', O)$ of t' and adds them to the list. This operation is repeated until the list is emptied. A list of "visited" triple patterns can

be used in order to avoid calling `applyRules` more than once for the same triple pattern. As each `applyRules`(t', O) call may generate some indirect relaxations of t', in a final step, edges associated with indirect relaxations are deleted using any standard method to compute the transitive reduction of a dag.

4.2 Computing the Relaxation Graph Incrementally

The naive procedure has a major drawback. In order to delete the indirect relaxations generated by `applyRules`(t, O), we need to call `applyRules` for all triples in the relaxation graph of t. This should be done even though the user is interested in relaxing t only one step further. In this section, we show how to transform the ontology O into a new ontology O' such that `applyRules`(t, O') only returns direct relaxations of t.

Now, let us return to the example of the previous subsection, where O is the ontology of Figure 5 (A). Recall that `applyRules`$((?X, \texttt{type}, B), O)$ generates the direct relaxation $(?X, \texttt{type}, C)$ and the indirect relaxation $(?X, \texttt{type}, D)$ of $(?X, \texttt{type}, B)$. The latter is generated using the triple $(B, \texttt{sc}, D)$ of O. We say that this is a *derivable triple* since it can be derived from other two triples in cl(O). Now, observe that if we delete $(B, \texttt{sc}, D)$ from O, we obtain a reduced version O' of O (which is logically equivalent to O) such that `applyRules`$((?X, \texttt{type}, B), O')$ only outputs direct relaxations. This example motivates us to delete the derivable triples of O since they produce indirect relaxations. The following proposition shows that this indeed is a good idea.

Proposition 4. *Let O be an ontology, o be a derivable triple in* cl(O) *and t, t' be triple patterns such that $t, o \vdash t'$. Then t' is an indirect relaxation of t (defined using O).*

From Proposition 4 we conclude that we should apply `applyRules` over the *reduction* of O instead of O. The reduction of O, denoted `red`(O), is the minimal ontology $O' \subseteq O$, such that cl(O') = cl(O). The reduction does not contain derivable triples and can be computed as follows (applying a rule in reverse means deleting the triple deduced by the rule): (i) compute cl(O); (ii) apply rule 4 in reverse until no longer applicable; and (iii) apply rules 1 and 3 in reverse until no longer applicable. In what follows, we assume that `red`(O) has been precomputed. Notice that, because every predicate in a triple in the ontology should be in $\{\texttt{type}, \texttt{dom}, \texttt{range}, \texttt{sp}, \texttt{sc}\}$, reverse rules 2, 5, and 6 are not needed to compute the reduction.

Since `red`(O) is logically equivalent to O, we obtain the same relaxations using `red`(O) and using O. The following proposition follows directly from Proposition 3.

Proposition 5. *Let O be an ontology and t be a triple pattern not in* cl(O). *Then all direct relaxations of t (defined using O) are in the set* `applyRules`$(t, \texttt{red}(O))$.

Unfortunately, `applyRules`$(t, \texttt{red}(O))$ may still return indirect relaxations, as the following example shows. Consider the reduction `red`(O) shown in Figure 5 (B), where O is the ontology of Figure 5 (A). Then, with the following instantiation of rule 5

$$(A, dom, C), (?X, A, ?Y) \vdash_5 (?X, \texttt{type}, C)$$

$\texttt{applyRules}((?X, A, ?Y), \texttt{red}(O))$ produces the relaxation $(?X, \texttt{type}, C)$ of $(?X, A, ?Y)$. However, this is an indirect relaxation, since we have that $(?X, A, ?Y) \prec (?X, \texttt{type}, B) \prec (?X, \texttt{type}, C)$. Fortunately, the following proposition shows that it is not difficult to detect the triples in the reduction that cause problems.

Proposition 6. *Let O be an ontology, o be a triple in $\texttt{red}(O)$ and t, t' be triple patterns such that $t, o \vdash t'$. Then t' is an indirect relaxation of t (defined using O) iff o can be derived by applying the rules of Figure 7 starting from $\mathrm{cl}(O)$.*

$$(e1)\ \frac{(b,\texttt{dom},c)\ \ (a,\texttt{sp},b)}{(a,\texttt{dom},c)} \qquad (e2)\ \frac{(b,\texttt{range},c)\ \ (a,\texttt{sp},b)}{(a,\texttt{range},c)}$$

$$(e3)\ \frac{(a,\texttt{dom},b)\ \ (b,\texttt{sc},c)}{(a,\texttt{dom},c)} \qquad (e4)\ \frac{(a,\texttt{range},b)\ \ (b,\texttt{sc},c)}{(a,\texttt{range},c)}$$

Fig. 7. Additional rules used to compute the extended reduction of an RDFS ontology

The proposition shows that we can avoid generating indirect relaxations by further reducing the ontology with the rules of Figure 7, which yields an extended reduction of an ontology O. The extended reduction, denoted $\texttt{extRed}(O)$, is defined as follows: (i) compute $\mathrm{cl}(O)$; (ii) apply the rules of Figure 7 in reverse until no longer applicable; (iii) apply rule 4 in reverse until no longer applicable; and (iv) apply rules 1 and 3 in reverse until no longer applicable.

As an example, Figure 5 (C) shows the extended reduction of the ontology of Figure 5 (A).

Now, observe that $\texttt{extRed}(O)$ may not be logically equivalent to O. However, the previous propositions show that we can still obtain all the direct relaxations of any triple pattern (defined using O) from $\texttt{extRed}(O)$. We are now ready to present the main result of this section.

Proposition 7. *Let O be an ontology and t be a triple pattern not in $\mathrm{cl}(O)$. Then $\texttt{applyRules}(t, \texttt{extRed}(O))$ is equal to the set of direct relaxations of t (defined using O).*

Figure 8 shows an algorithm that computes the relaxation graph of a triple pattern incrementally. We assume that the extended reduction $\texttt{extRed}(O)$ has been previously computed and stored. The variable `Frontier` keeps a list of triple patterns. The variables V and E keep the nodes and edges of the relaxation graph, respectively. Notice that, because of Proposition 7, the algorithm does not produce indirect relaxations.

4.3 Complexity

We now give a bound on the size of the relaxation graph.

Proposition 8. *Let t be a triple pattern and O be an ontology. The relaxation graph of t (using the ontology O) has $O(|\texttt{red}(O)|^2)$ triples.*

Input: A triple pattern t and the extended reduction $\texttt{extRed}(O)$ of an ontology O.
Output: The relaxation graph of t (using O).

$\texttt{Frontier} := \langle t \rangle$
$V := \{t\}, E := \emptyset$
While (**Frontier** is non-empty)
    Delete first element u from **Frontier**
    $U := \texttt{applyRules}(u, \texttt{extRed}(O))$
    $V := V \cup U$
    $E := E \cup \{(u, u') : u' \in U\}$
    Add the triple patterns in U to **Frontier**
Return (V, E)

Fig. 8. Algorithm that computes the relaxation graph of a triple pattern

From Proposition 9, it follows that the relaxation graph of a query has $O(|\texttt{red}(O)|^{2n})$ nodes, where n is the number of triple patterns inside `RELAX` clauses in the query.

Proposition 9. *Let t be a triple pattern and O be an ontology. (i) Computing the direct relaxations of t takes $O(|\texttt{red}(O)|)$ steps. (ii) Computing the relaxation graph of t takes $O(|\texttt{red}(O)|^3)$ steps.*

5 Computing the Relaxed Answer

In this section, we study the problem of computing the relaxed answer of a query. We propose an algorithm that incrementally generates matchings from a query to an RDF graph and also ranks tuples in the answer.

Our query processing algorithm works by adapting the RDQL query processing scheme provided by Jena [21] to the processing of successive relaxations of a query. We assume the simplest storage scheme provided by Jena, in which the RDF triples are stored in a single table, called the *statement table*. The Jena query processing approach is to convert an RDF query into a pipeline of "find patterns" connected by join variables. Each triple pattern (find pattern in Jena's terminology) can be evaluated by a single SQL select query over the statement table. We formalize this with an operator called `find` that receives a triple pattern t and a statement table G and returns all matchings from t to the table.

In Section 5.1 we present our algorithm for efficiently computing the relaxed answer of a query and we prove its correctness. In Sections 5.2 and 5.3 we illustrate two examples of the execution of the algorithm. Finally, in Section 5.4, we study the complexity of the algorithm.

5.1 Algorithm

In what follows, Q is the query whose relaxed answer we intend to compute, and Q' is an arbitrary query in the relaxation graph of Q. We have that $H =$

$\texttt{Head}(Q) = \texttt{Head}(Q')$. For the sake of simplicity, we assume that each triple pattern in the body of Q is inside a `RELAX` clause. We assume that $\texttt{Body}(Q) = \{t_1, \ldots, t_n\}$, and $\texttt{Body}(Q') = \{t'_1, \ldots, \ldots, t'_n\}$. We also fix the statement table G we are querying. The answer of Q' can be computed by processing (in a pipelined fashion) a view, denoted $V_{Q'}$, defined by the following expression:

$$\pi_H(\texttt{find}(t'_1, G) \bowtie \ldots \bowtie \texttt{find}(t'_n, G)),$$

where π is the standard projection operator and $\bowtie$ is the natural join on variables shared by triple patterns. The answer of Q can be computed by a naive algorithm that traverses the relaxation graph of Q upwards, and in each step of the traversal, builds a view $V_{Q'}$, computes it, and returns those tuples which were not returned in previous steps.

Next, we propose an algorithm that avoids the redundant processing of tuples that arises with this naive approach. We define $\texttt{deltaFind}(t'_i, G)$ as the set containing triples $p \in G$ such that t'_i matches p, and no triple pattern directly below t'_i in the relaxation graph of t_i, matches p. The set $\texttt{deltaFind}(t'_i, G)$ can be computed similarly to $\texttt{find}(t'_i, G)$ by filtering triples from the statement table. We define a *delta view* for Q', denoted $\Delta_{Q'}$, as the following expression:

$$\pi_H(\texttt{deltaFind}(t'_1, G) \bowtie \ldots \bowtie \texttt{deltaFind}(t'_n, G)).$$

The following proposition shows that new answers (Section 3.5) correspond to delta views.

Proposition 10. *Let Q be a query and G be a RDF graph. For each query Q' in the relaxation graph of Q, (i)* $\texttt{ans}(Q', G) = \bigcup_{Q_i : Q_i \leq_Q Q'} \Delta_{Q_i}(G)$*, and (ii)* $\texttt{newAnswer}(Q', G) = \Delta_{Q'}(G)$.

The algorithm we propose (Figure 9), called `RelaxEval`, performs a breadth-first traversal of the relaxation graph of Q, building and processing each delta view $\Delta_{Q'}$ in each step of the traversal. The function *level* returns the level of a triple pattern t'_i in the relaxation graph R_i of t_i. Line 3(a) outputs the new answer of each query at level k. In order to find the queries at level k of the relaxation graph, the algorithm applies the following property. The queries Q' (defined by the join expression in Line 3 (a)) that belong to the level k of the relaxation graph of Q are those satisfying $\sum_i level(t'_i, R_i) = k$.

We next prove the correctness of `RelaxEval`.

Proposition 11. *The algorithm* `RelaxEval` *returns its tuples in ranked order.*

5.2 Example

We next illustrate the algorithm with the following query Q (that we also presented in the example of Section 3.3):

$$?X \leftarrow \{\texttt{RELAX}\{(?X, \texttt{type}, \mathit{Publication})\}, \texttt{RELAX}\{(\mathit{JohnRobert}, \mathit{editorOf}, ?X)\}\}.$$

Algorithm `RelaxEval`
Input: a query Q (interpreted over an ontology O), where $\texttt{Body}(Q) = \{t_1, \ldots, t_n\}$, a statement table G, and an integer *maxLevel*.
Output: the set of tuples $\texttt{ans}_{\texttt{relax}}(Q, G, maxLevel)$, where new answers are returned successively at each level of the relaxation graph.

1. $k := 0$, *stillMore* := *true*
2. For each triple pattern $t_i \in \texttt{Body}(Q)$, compute the relaxation graph R_i of t_i up to level *maxLevel*.
3. While ($k \leq maxLevel$ and *stillMore*) do
   (a) For each combination $t'_1 \in R_1, \ldots, t'_n \in R_n$ such that $\sum_i level(t'_i, R_i) = k$ do output $\pi_H(\texttt{deltaFind}(t'_1, G) \bowtie \ldots \bowtie \texttt{deltaFind}(t'_n, G))$
   (b) $k := k + 1$
   (c) *stillMore* := exist nodes $t'_1 \in R_1, \ldots, t'_n \in R_n$ such that $\sum_i level(t'_i, R_i) = k$

Fig. 9. Algorithm that computes the relaxed answer of a query

For simplicity, we consider subgraphs of the relaxation graphs of triple patterns in the query, shown in Figure 4 (A). Figure 4 (B) shows the relaxation graph of the query, which is obtained by combining the graphs of Figure 4 (A).

We assume the query is interpreted in the context of an ontology O, which consists of the subgraph with edges `sc` and `type` of the ontology of Figure 1. The

Statement table G		
Subject	Predicate	Object
a	`type`	*Publication*
b	`type`	*WebPage*
c	`type`	*Publication*
d	`type`	*WebPage*
JohnRobert	*editorOf*	*a*
JohnRobert	*editorOf*	*b*
JohnRobert	*authorOf*	*c*
JohnRobert	*authorOf*	*d*

(A)

Statement table for cl(G, O)		
Subject	Predicate	Object
a	`type`	*Publication*
b	`type`	*WebPage*
c	`type`	*Publication*
d	`type`	*WebPage*
a	`type`	*Document*
b	`type`	*Document*
c	`type`	*Document*
d	`type`	*Document*
JohnRobert	*editorOf*	*a*
JohnRobert	*editorOf*	*b*
JohnRobert	*authorOf*	*c*
JohnRobert	*authorOf*	*d*
JohnRobert	*contributorOf*	*a*
JohnRobert	*contributorOf*	*b*
JohnRobert	*contributorOf*	*c*
JohnRobert	*contributorOf*	*d*

(B)

Fig. 10. (A) Statement Table G. (B) A statement table containing cl(G, O), where O is the subgraph of the ontology of Figure 1 that includes only the `sc` and `type` subgraphs.

query is interpreted over the statement table G given in Figure 10 (A), whose closure $\mathrm{cl}(G, O)$ is given in Figure 10 (B).

Figure 11 (A) shows the answers of queries in the relaxation graph of Q and Figure 11 (B) shows the answers of delta views. An answer is a set of tuples; since the query at hand has a single head variable, each tuple is a single element in our example. Figure 11 (C) shows the answer returned by `RelaxEval` at levels $0, 1$ and 2.

Notice that Proposition 10 (i) and (ii) hold. For instance, for query Q_3, we have

$$\texttt{ans}(Q_3, G) = \Delta_{Q_3} \cup (\Delta_{Q_1} \cup \Delta_{Q_2} \cup \Delta_{Q_0}),$$

and we also have

$$\Delta_{Q_3}(G) = \texttt{ans}(Q_3, G) - (\texttt{ans}(Q_1, G) \cup \texttt{ans}(Q_2, G) \cup \texttt{ans}(Q_0, G)).$$

We now illustrate how `RelaxEval` computes the delta view

$$\Delta_{Q_3}(G) := \pi_{?X}(\texttt{deltaFind}((?X, \texttt{type}, \mathit{Document}), G) \bowtie \texttt{deltaFind}((\mathit{JohnRobert}, \mathit{contributorOf}, ?X), G)).$$

Here, $\texttt{deltaFind}((?X, \texttt{type}, \mathit{Document}), G)$ finds all matchings μ from ($?X$, `type`, *Document*) to $\mathrm{cl}(G, 0)$ such that μ is not a matching from ($?X$, `type`, *Publication*) to $\mathrm{cl}(G, O)$ (because $(?X, \texttt{type}, \mathit{Publication})$ is the only triple pattern directly below $(?X, \texttt{type}, \mathit{Document})$ in the relaxation graph of ($?X$, `type`, *Publication*)). Therefore, $\texttt{deltaFind}((?X$, `type`, *Document*$), G)$ returns the following table:

$?X$	`type`	*Document*
b	`type`	*Document*
d	`type`	*Document*

Similarly, $\texttt{deltaFind}((\mathit{JohnRobert}, \mathit{contributorOf}, ?X), G)$ computes the following table:

JohnRobert	*contributorOf*	$?X$
JohnRobert	*contributorOf*	c
JohnRobert	*contributorOf*	d

Therefore, $\Delta_{Q_3}(G) = \{d\}$.

5.3 A Further Example — Heterogeneous Database Integration

We now discuss how our algorithm for incrementally computing the relaxed answer of a query might be applied in a heterogeneous data integration setting, specifically in the integration and querying of multiple heterogeneous proteomic data resources.

Answers of Relaxed Queries	
Relaxed Query	Answer
Q_0	$\{a\}$
Q_1	$\{a, b\}$
Q_2	$\{a, c\}$
Q_3	$\{a, b, c, d\}$

(A)

Answers of Delta Views	
Delta view	Answer
Δ_{Q_0}	$\{a\}$
Δ_{Q_1}	$\{b\}$
Δ_{Q_2}	$\{c\}$
Δ_{Q_3}	$\{d\}$

(B)

Answers of `RelaxEval`	
Level	Answer
0	$\{a\}$
1	$\{b, c\}$
2	$\{d\}$

(C)

Fig. 11. (A) Answers of relaxed queries until level 2. (B) Delta views for the relaxed queries until level 2. (C) Tuples returned by `RelaxEval` per level until level 2.

Proteomic data resources are rapidly being developed globally, with the emergence of affordable, reliable methods to study the proteome. The *In Silico Proteome Integrated Data Environment Resource* (ISPIDER) project[4] is developing an integrated platform of proteome-related resources, using existing standards from proteomics, bioinformatics and e-Science. The integration of such resources is beneficial for a number of reasons. First, having access to more data leads to more reliable analyses; for example, performing protein identifications over an integrated resource reduces the chances of false negatives. Second, bringing together resources containing different but closely related data increases the breadth of information the biologist has access to. Third, the integration of these resources, as opposed to merely providing a common interface for accessing them, enables data from a range of experiments, tissues, or different cell states to be brought together in a form which may be analysed by a biologist in spite of the widely varying coverage and underlying technology of each resource.

In the ISPIDER project, we have developed an architecture which supports the combined use of Grid data access, Grid distributed querying and data integration software tools. This architecture allows us to develop an integrated global schema over heterogeneous resources and to support distributed queries posed over such a global schema. Reference [22] reports on our initial results from the integration of three distributed, autonomous proteomics resources, all of which contain information about protein and peptide identification: gpmDB[5], Pedro[6] and PepSeeker[7].

As reported in [22], building an integrated global schema over such heterogeneous proteomics resources is a lengthy and complex process. Indeed, so far, we have not performed a full integration of these three databases, but only a limited integration such that the global schema captures enough information for answering common proteomics questions. Moreover, some of the resource schemas are still under development and enhancement, which requires ongoing modification to our integration mappings and global schema.

An alternative approach, therefore, would be to undertake a "light-weight" integration of these resources, producing a global ontology that captures the

[4] See http://www.ispider.manchester.ac.uk
[5] See http://gpmdb.thegpm.org
[6] See http://pedrodb.man.ac.uk:8080/pedrodb
[7] See http://www.nwsr.manchester.ac.uk/pepseeker

classes and properties of the individual resources as well as their common concepts (i.e. super-classes and super-properties of the local ontology classes and properties), and to use our query processing algorithm to incrementally relax and compute the answers to queries over this global ontology.

For example, in the global ontology, there may be

- classes *PedroPeptide* and *PedroProtein*
- and properties
  - *PedroPeptideSequence*, with domain *PedroPeptide* and range Literal,
  - *PedroAligns*, with domain *PedroPeptide* and range *PedroProtein*,
  - *PedroAccessionNumber*, with domain *PedroProtein* and range Literal,

arising from the Pedro resource, which is based at Manchester (in Mass Spectrometry experiments, several Peptides result from the identification process; each Peptide aligns against a set of Proteins; a Protein is characterized by a textual description, an accession number, the predicted mass of the protein, the organism in which it is to be found, etc.).

There may be a similar set of classes and properties arising from the PepSeeker resource, also at Manchester:

- classes *PepPeptide* and *PepProtein*
- and properties
  - *PepPeptideSequence*, with domain *PepPeptide* and range Literal,
  - *PepAligns*, with domain *PepPeptide* and range *PepProtein*,
  - *PepAccessionNumber*, with domain *PepProtein* and range Literal,

In the global ontology there may be superclasses and superproperties of the above which collectively represent the information in the Manchester resources:

- superclasses *ManchPeptide* and *ManchProtein*
- and superproperties
  - *ManchPeptideSequence*, with domain *ManchPeptide* and range Literal,
  - *ManchAligns*, with domain *ManchPeptide* and range *ManchProtein*,
  - *ManchAccessionNumber*, with domain *ManchProtein* and range Literal,

This fragment of the ontology is shown in Figure 12.

We may also have properties and classes in the global ontology, arising from the publicly available gpmDB resources:

- classes: *gpmPeptide* and *gpmProtein*
- properties: *gpmPeptideSequence*, *gpmAligns* and *gpmAccessionNumber*

Finally, there may be the following classes and properties that are supeclasses or superproperties of the corresponding Manchester and gpmDB classes or properties, and that collectively represent the information in all three resources:

- superclasses: *Peptide* and *Protein*
- superproperties: *PeptideSequence*, *Aligns* and *AccessionNumber*

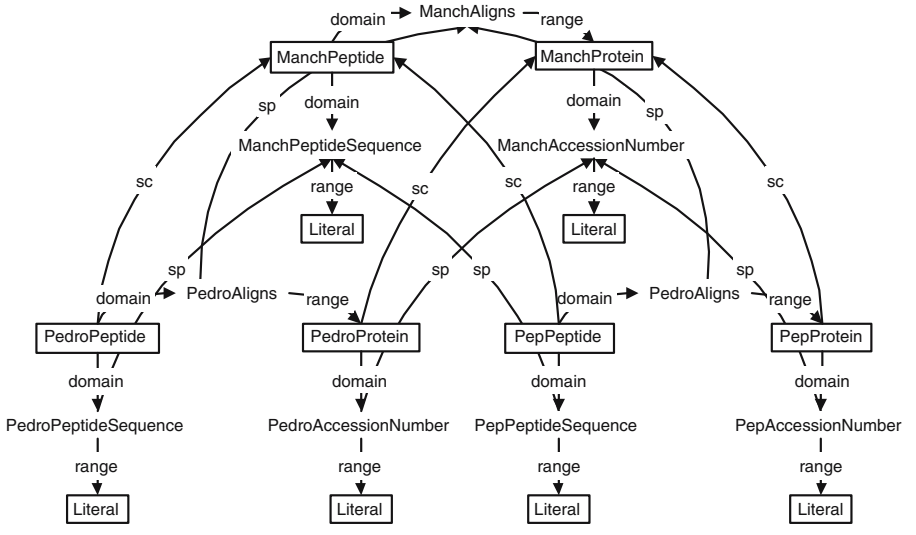

Fig. 12. Part of the Proteomics Resources Ontology

Consider now the following query posed over the global ontology by a user who is familiar with the Pedro resource:

```
?Y, ?Z <- {RELAX{(?X,PedroPeptideSequence,"ATLITFLCDR")},
           RELAX{(?X,PedroAligns,?Y)},
           RELAX{(?Y,PedroAccessionNumber,?Z)}}
```

In its non-relaxed form, this query will return the identifiers and accession numbers of proteins that have been identified within the Pedro resource as a result of experiments that have yielded the peptide "ATLITFLCDR". Such a query allows a scientist working with a protein sequence to ask if this peptide has been seen before in other proteomics experiments.

A first level of relaxation of the three literals in this query according to the `sp` subgraph, will result in the following relaxed query,

```
?Y, ?Z <- {RELAX{(?X,ManchPeptideSequence,"ATLITFLCDR")},
           RELAX{(?X,ManchAligns,?Y)},
           RELAX{(?Y,ManchAccessionNumber,?Z)}}
```

which will expand out the result set to include results also from the other Manchester resource, Pepseeker, without the Pedro user needing to have detailed knowledge of the schema of that resource.

A further level of relaxation of the three literals in the query according to the `sp` subgraph, will result in the following relaxed query,

```
?Y, ?Z <- {RELAX{(?X,PeptideSequence,"ATLITFLCDR")},
           RELAX{(?X,Aligns,?Y)},
           RELAX{(?Y,AccessionNumber,?Z)}}
```

which will now expand out the result set to include results also from the gpmDB resource, again without the Pedro user needing to have detailed knowledge of the schema of that resource.

In contrast therefore to the approach discussed in [22], in which users must pose queries against an integrated global schema, the light-weight integration and relaxed querying approach that we have outlined here would allow a more incremental construction of query results, a more exploratory approach to query answering, and also less knowledge of the global resources by users.

5.4 Complexity

The complexity of `RelaxEval` is given by the following proposition.

Proposition 12. *Let Q be a query, O be an ontology and G an RDF graph. Then* $\texttt{RelaxEval}(Q, G, k)$ *runs in time* $O(m^{2n}|G|^n)$*, where m is the number of triples in* $\texttt{red}(O)$*, and* $n = |\texttt{Body}(Q)|$.

The above proposition shows that the algorithm has exponential complexity, however its complexity is polynomial in the size of the data queried for a fixed query Q (data complexity). In addition, the answer is generated incrementally and hence the processing can be halted at any level in the relaxation graph. The number of triples in $\texttt{red}(O)$ provides an upper bound for k, the number of levels in the evaluation.

An improvement to the algorithm would be to process several delta views at the same time in an integrated pipelined fashion. In practice, we can improve query processing performance by further caching the results of $\texttt{deltaFind}(t, G)$ for all triple patterns t that occur more than once in the query relaxation graph (such duplicate occurrences can be detected as the relaxation graphs of the individual triple patterns in the original query are being constructed).

6 Introducing Simple Relaxations

In this section, we extend the relaxation framework to consider simple relaxations that is, relaxations that replace terms of the original triple pattern with variables. In Section 6.1, we formalize simple relaxation and show how the relaxation graph can be extended with simple relaxations. In Section 6.2, we show the additional types of relaxation now captured by our framework.

6.1 Notion of Simple Relaxation

An important restriction we place in our framework is to prevent simple relaxations replacing variables of the original query. This is because such variables are needed to export results and join triple patterns in the relaxed queries. It is important to note, however, that (as we will show in the next section) this restriction does not limit the ability of our framework to relax join dependencies.

In the light of the above, we call the variables of the original query *fixed variables*, and define the notion of map that preserves such variables, along with literals and IRIs. A *map* from a triple pattern $t_1 = (a, b, c)$ to a triple pattern $t_2 = (d, e, f)$ is a function μ from terms of t_1 to terms of t_2, preserving IRIs, literals, and fixed variables, such that $(\mu(a), \mu(b), \mu(c))) = (d, e, f)$. We say that two triple patterns t_1 and t_2 are isomorphic if there are maps from t_1 to t_2 and from t_2 to t_1. Now, we define simple relaxation.

Definition 5 (Simple Relaxation). *If t_1, t_2 are triple patterns, then $t_1 \leq_{\texttt{simple}} t_2$ if there is a map from t_2 to t_1.*

As an example, assuming a unique fixed variable $?X$, we have $(?X, \texttt{type}, \mathit{Article}) \leq_{\texttt{simple}} (?X, \texttt{type}, ?Z)$ and $(?X, \texttt{type}, \mathit{Article}) \leq_{\texttt{simple}} (?X, ?W, \mathit{Article})$, among other simple relaxations. It is not the case that $(?X, \texttt{type}, \mathit{Article}) \leq_{\texttt{simple}} (?U, \texttt{type}, ?Z)$, since the fixed variable $?X$ is replaced.

The following proposition confirms a desired property of the simple relaxation relation.

Proposition 13. *The simple relaxation relation $\leq_{\texttt{simple}}$ is a partial order up to triple pattern isomorphism.*

Similarly to Section 3.2, we can define the notion of relaxation graph of a triple pattern t. It is enough to define the direct simple relaxation relation $\prec_{\texttt{simple}}$ (transitive and reflexive reduction of $\leq_{\texttt{simple}}$ up to isomorphism), and the relation $\prec_{\texttt{simple},t}$ (simple relaxation "above" t). The simple relaxation graph of t is the graph induced by $\prec_{\texttt{simple},t}$. This graph is unique up to triple pattern isomorphism. In order to obtain a clean representation of it, without loss of generality, we may assume that each non-fixed variable does not appear in more than one triple pattern.

Also notice that the simple relaxation graph of a triple pattern can be easily computed: we just need to iteratively replace terms by variables in triple patterns, taking care not to generate isomorphic triples and indirect relaxations.

The notions of relaxation graph of a triple pattern (and hence of a query) introduced in Section 3.2 can be naturally generalized to include simple relaxations in different ways. Here we sketch one possible, yet simple, extension. We add on the top of each triple pattern t in the original relaxation graph the simple relaxation graph of t, and then delete indirect edges.

As an example, consider the ontology O of Figure 1. Figure 13 (B) shows the relaxation graph of $(\mathit{JohnRobert}, \mathit{editorOf}, ?X)$ (Figure 3) extended with simple relaxations. The non-fixed variables are $?U1, \ldots, ?U10$.

Finally, we just remark that the query processing algorithm of Section 5 can be applied without any modification to the extended version of the relaxation graph as well.

6.2 Types of Simple Relaxation

The following types of relaxation can be captured by simple relaxation.

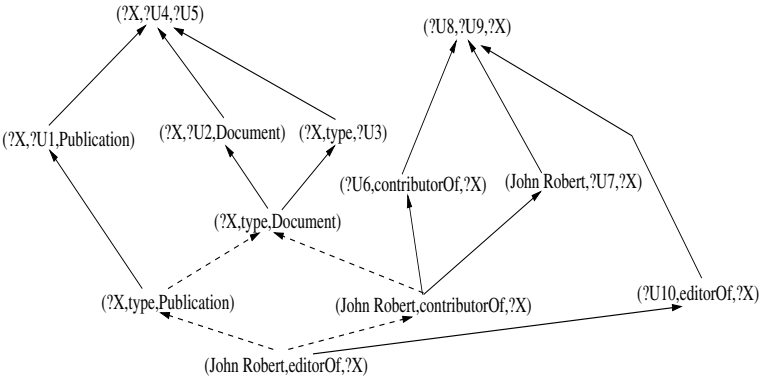

Fig. 13. Relaxation graph of the triple pattern $(JohnRobert, editorOf, ?X)$ considering simple relaxations. The simple relaxations are shown with solid arrows.

1. Dropping triple patterns. We can model the dropping of triple patterns by introducing an "empty" triple pattern, which can be regarded as a "true" condition to which any triple pattern relaxes. In this form, relaxation generalizes the use of the `OPTIONAL` clause within the conjunctive fragment of SPARQL.
2. Constant relaxation: replacing a constant with a variable in a triple pattern. This can be further classified according to whether the variable replaces a property or a subject/object constant.
3. Breaking join dependencies: generating new variable names for a variable that appears in multiple triple patterns. In order to model this type of relaxation, we first transform queries by applying variable substitution. If a variable $?X$ appears $n > 1$ times in a query Q we replace each occurrence with a different variable and add triple patterns $(?X_i, \texttt{equal}, ?X_j)$ for each pair of new variables $?X_i, ?X_j$ introduced. The predicate `equal` represents equality. Each of the equality clauses in a query can now also be subject to relaxation.

7 Related Work

Query languages based on regular expressions provide a form of flexible querying. The G^+ query language by Cruz et al. [6] proposes graph patterns where edges are annotated with regular expressions over labels. In this form, each graph pattern represents a set of more basic graph patterns, and therefore, a query extracts matchings that relate to its body in a variety of ways. This work considers queries over directed labeled graphs.

Kanza and Sagiv [14] propose a form of flexible querying based on a notion of homeomorphism between the query and the graph. Their data model is a simplified form of the Object Exchange Model (OEM).

Bernstein and Kiefer [1] incorporate similarity joins into the RDQL query language. This is done by allowing sets of variables in an RDQL query to be declared as *imprecise*. Bindings for these variables are then compared based on a specified similarity measure, such as edit distance.

Stuckenschmidt and van Harmelen [20] consider conjunctive queries over a terminological knowledge base that includes class, relation and object definitions. They also use query containment as a way of viewing query approximations, but are concerned about evaluating less complex queries first, so that the original query is evaluated last. They use a query graph to decide which conjuncts from the original query should be successively added to the approximate query. This is analogous to SPARQL queries in which every conjunct is optional.

Bulskov et al. [4] consider the language ONTOLOG which allows compound concepts to be formed from atomic concepts attributed with semantic relations. They define a similarity measure between concepts based on subsumption in a hierarchy of concepts. This gives rise to a fuzzy set of concepts similar to a given concept. They also introduce specialization/generalization operators into a query language that allow specializations or generalizations of concepts to be returned. They admit that combining this with similarity may make answers confusing.

In a series of papers, e.g. [18,19], Stojanovic and others have studied the problem of query refinement in information retrieval, where users tend to pose initial queries that are too short to fulfull their needs. The techniques proposed use ontologies associated with the information to analyse "amibiguities" in the user's queries as well as users' preferences in order to suggest incremental refinements to the user. For example, [19] uses a form of subsumption between queries which generates a lattice of query refinements. A form of ranking, based on user modelling and monitoring, is also provided. However, the refinements considered are, in fact, specialisations of a query, rather than generalisations as in our case. Generalisation is considered as one form of query refinement in [18]. The ontologies used in all cases, however, are not based on RDF/S.

A recent paper by Dolog et at. considers relaxing over-constrained queries on RDF [7]. The paper proposes a rewriting technique based on domain knowledge and user preferences, although these are not encoded using RDFS. The implementation of rewriting is performed using event-condition-action rules, for which the authors state that termination of execution still needs to be thoroughly investigated.

8 Concluding Remarks

Despite being a relatively unexplored technique in the semantic Web, query relaxation may have an important role in improving RDF data access. One motivation for this technique is for querying data where there is a lack of understanding of the ontology that underlies the data. Another application is the extraction of objects with heterogeneous sets of properties because the data is incomplete or has irregular structure. As an example, a relaxed query can retrieve the

properties that are applicable to each resource among a set of resources having different properties. Query relaxation can also make it possible to retrieve data that satisfies the query conditions with different degrees of exactitude. Another application area where this facility could be useful is the discovery of semantic web services.

There are several areas for future work. One is the introduction of relaxation into general SPARQL queries, including disjunctions and optionals. This should also involve a generalization of the `RELAX` clause so that it can be applied to entire graph patterns instead of single triple patterns. Another important issue for future work is the design, implementation and empirical evaluation of algorithms for computing relaxed answers. Finally, the graph-like nature of RDF provides additional richness for a query relaxation framework, which can be exploited in future work. For example, join dependencies between triple patterns of the query can be relaxed to connectivity relationships in RDF graphs.

Acknowledgement

We gratefully acknowledge the contribution of Lucas Zamboulis in the formulation of the example in Section 5.3.

References

1. Bernstein, A., Kiefer, C.: Imprecise RDQL: Towards generic retrieval in ontologies using similarity joins. In: SAC/SIGAPP. 21th Annual ACM Symposium on Applied Computing, Dijon, France (2006)
2. Brickley, D., Guha, R.V. (eds.): RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation (February 10, 2004)
3. Broekstra, J.: SeRQL: Sesame RDF query language. In: Ehrig, M., et al. (eds.) SWAP Deliverable 3.2 Method Design, pp. 55+68 (2003), `http://swap.semanticweb.org/public/Publications/swap-d3.2.pdf`
4. Bulskov, H., Knappe, R., Andreasen, T.: On querying ontologies and databases. In: 6th International Conference on Flexible Query Answering Systems, pp. 191–202 (2004)
5. Clark, K.G. (ed.): RDF Data Access Use Cases and Requirements, W3C Working Draft (March 25, 2005)
6. Cruz, I.F., Mendelzon, A.O., Wood, P.T.: A graphical query language supporting recursion. In: ACM SIGMOD International Conference on Management of Data, pp. 323–330 (1987)
7. Dolog, P., Stuckenschmidt, H., Wache, H.: Robust query processing for personalized information access on the semantic web. In: 7th International Conference on Flexible Query Answering Systems, pp. 343–355 (2006)
8. Fikes, R., Hayes, P.J., Horrocks, I.: OWL-QL - a language for deductive query answering on the semantic web. J. Web Sem. 2(1), 19–29 (2004)
9. Gaasterland, T., Godfrey, P., Minker, J.: Relaxation as a platform for cooperative answering. J. Intell. Inf. Syst. 1(3/4), 293–321 (1992)
10. Gutierrez, C., Hurtado, C., Mendelzon, A.O.: Foundations of semantic web databases. In: 23rd Symposium on Principles of Database Systems, pp. 95–106 (2004)

11. Haase, P., Broekstra, J., Eberhart, A., Volz, R.: A comparison of RDF query languages. In: International Semantic Web Conference (2004)
12. Hayes, P.: RDF Semantics, W3C Recommendation (February 10, 2004)
13. Hurtado, C., Poulovassilis, A., Wood, P.T.: A relaxed approach to RDF querying. In: Proceedings of the 5th International Semantic Web Conference, Athens, GA, USA, pp. 314–328 (2007)
14. Kanza, Y., Sagiv, Y.: Flexible queries over semistructured data. In: 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 40–51 (2001)
15. Manola, F., Miller, E. (eds.): RDF Primer, W3C Recommendation (February 10, 2004)
16. Nerode, A., Shore, R.: Logic for Applications. Springer, Netherlands (1998)
17. Prud'hommeaux, E., Seaborne, A. (eds.): SPARQL Query Language for RDF, W3C Candidate Recommendation (April 6, 2006)
18. Stojanovic, N.: Information-need driven query refinement. In: Proceedings of the IEEE/WIC International Conference on Web Intelligence, pp. 388–395 (2003)
19. Stojanovic, N., Stojanovic, L.: A logic-based approach for query refinement in ontology-based information retrieval systems. In: 16th IEEE International Conference on Tools with Artificial Intelligence, pp. 450–457 (2004)
20. Stuckenschmidt, H., van Harmelen, F.: Approximating terminological queries. In: 5th International Conference on Flexible Query Answering Systems, pp. 329–343 (2002)
21. Wilkinson, K., Sayers, C., Kuno, H., Reynolds, D.: Efficient RDF storage and retrieval in Jena. In: Proceedings of VLDB Workshop on Semantic Web and Databases (2003)
22. Zamboulis, L., Fan, H., Khalid, B., Siepen, J.A., Jones, A., Martin, N.J., Poulovassilis, A., Hubbard, S.J., Embury, S.M., Paton, N.W.: Data access and integration in the ISPIDER proteomics grid. In: Leser, U., Naumann, F., Eckman, B. (eds.) DILS 2006. LNCS (LNBI), vol. 4075, pp. 3–18. Springer, Heidelberg (2006)

A Proofs

Proposition 1. Let $\leq$ be defined using an ontology O, and t_1, t_2 be triple patterns such that $t_1, t_2 \notin \mathrm{cl}(O)$ and $\mathtt{var}(t_1) = \mathtt{var}(t_2)$. Then $t_1 \leq t_2$ if and only if $t_2 \in \mathrm{cl}(O \cup \{t_1\})$.

Proof of Proposition 1. Follows directly from the definition of relaxation. □

Proposition 2. Let $\leq$ be defined using an ontology O. Then $\leq$ is a partial order if and only if O is acyclic.

Proof of Proposition 2. (Only If) We prove the contrapositive, that is, if O is cyclic then $\leq$ is not a partial order. We prove it for the case where the subgraph $O_{\mathtt{sc}}$ of O induced by $\mathtt{sc}$ is cyclic (the proof for a cycle in the subproperty graph is similar). It is enough to prove that there exist triple patterns t_a, t_b such that $t_a \leq t_b$ and $t_b \leq t_a$. Now consider the following cycle in $O_{\mathtt{sc}}$: $(e_1, \mathtt{sc}, e_2), (e_2, \mathtt{sc}, e_3), \ldots, (e_{n-1}, \mathtt{sc}, e_1)$ and the following triple patterns $t_a = (c, \mathtt{type}, e_1)$ and $t_b = (c, \mathtt{type}, e_2)$. Because of rules 3 and 4, it can be easily

verified that there exists a derivation from t_a to t_b and another derivation from t_b to t_a. Hence $t_a \leq t_b$ and $t_b \leq t_a$.

(If) Assume that O is acyclic. It can be easily verified that $\leq$ is transitive and reflexive. Therefore, it remains to prove that it is antisymmetric. Now, assume that there exist triple patterns t_a, t_b, $t_a \neq t_b$, such that $t_a \leq t_b$ and $t_b \leq t_a$. We will proceed by cases, where each case is a possible form that t_a may take in order to instantiate at least one rule. By U, V, W we denote a IRI, variable or a literal, and by a, b, c we denote IRIs and literals. Notice that t_a cannot be a `dom`-triple or `range`-triple, because in this case, the only rules that can be instantiated are rules 5 or 6, and they require the existence of a plain triple in the ontology, which is not allowed. We use the notion of linear derivation from the proof of Proposition 3.

We prove by cases.

- t_a is a `type`-triple. In this case, rule 4 is the only rule that t_a can instantiate, hence t_b is also a `type`-triple. Let $t_a = (U, \texttt{type}, b)$, then the there is a linear derivation of the form $(U, \texttt{type}, a), (a, \texttt{sc}, e_1) \vdash_4 (U, \texttt{type}, e_1), (e_1, \texttt{sc}, e_2) \vdash_4 (U, \texttt{type}, e_2), (e_2, \texttt{sc}, e_3) \vdash_4 \ldots (U, \texttt{type}, e_n), (e_n, \texttt{sc}, b) \vdash_4 (U, \texttt{type}, b)$. That is $t_b = (U, \texttt{type}, b)$. Therefore, there must be a path from a to b in $O_{\texttt{sc}}$. By a similar argument, we prove the existence of a path from b to a in $O_{\texttt{sc}}$, contradicting that $O_{\texttt{sc}}$ is acyclic.
- t_a is an `sc`-triple. In this case, rules 3 and 4 are the only rules that can be instantiated by t_a. Hence t_b is either a `type`-triple or an `sc`-triple. If the former holds, then there is no derivation from t_b to t_a, a contradiction. Therefore, t_b is a `sc`-triple. Let $t_a = (a, \texttt{sc}, b)$ and let $t_b = (c, \texttt{sc}, d)$. In this case the internal nodes of the derivation graph are `sc`-triples. It can be easily verified that a path exists in $O_{\texttt{sc}} \cup \{(a, \texttt{sc}, b)\}$ from c to d that contains an edge $(a, \texttt{sc}, b)$. Similarly, because there is a derivation from t_b to t_a, there must exist a path in $O_{\texttt{sc}} \cup \{(c, \texttt{sc}, d)\}$ from a to b that contains an edge $(c, \texttt{sc}, d)$. It can be checked that $O_{\texttt{sc}}$ has at least one cycle, a contradiction.
- t_a is an `sp`-triple. Only rule 1 can apply because plain triples are not allowed in the ontology. Hence t_b is an `sp`-triple. By a similar argument as the proof of the previous case we prove the existence of a cycle in $O_{\texttt{sp}}$, yielding a contradiction.
- t_a is a plain triple. In this case, the only rules in the derivation graph from t_a to t_b are rules 2, 4, 5, and 6. These rules only yield `dom`-triples, `range`-triples, `type`-triples and plain triples. However, among these, a plain triple can be derived only from a plain-triple. Hence t_b is a plain triple, and the derivation graph has only instances of rule 2. Without loss of generality, let $t_a = (a, p, b)$ and $t_b = (a, q, b)$. Using a similar argument than the one used in the previous cases, we reach the conclusion that there are paths from p to q and from q to p in $O_{\texttt{sp}}$, a contradiction. □

Proposition 3. Let t_a, t_b be triple patterns not in $\text{cl}(O)$ such that $\texttt{var}(t_a) = \texttt{var}(t_b)$. If $t_a \prec t_b$ then there exists a triple $o \in \text{cl}(O)$ such that $t_a, o \vdash t_b$.

Proof of Proposition 3. We define that a *derivation* from a graph G (which may have triple patterns) to a triple pattern b_n is a sequence $a_1, a_2 \vdash b_3$; $a_4, a_5 \vdash b_6$; $a_7, a_8 \vdash b_9$; $\ldots a_{n-2}, a_{n-1} \vdash b_n$, where each $a_i, a_{i+1} \vdash b_{i+2}$ is an instantiation of a rule and a_i, a_{i+1} either belong to G or appear as the consequent b_j of some rule where $j < i$.

A derivation is said to be *linear* if each $b_j = a_{j+1}$ and a_{j+2} belongs to G. This notion is analogous to the notion of linear proof (e.g., linear proofs in Prolog's resolution). The intuition here is that when choosing two triples to combine in a derivation, always make one be the result of a previous derived triple and the other a triple from the original graph. A linear derivation from G to b_n can be abbreviated as follows: $a_1, a_2 \vdash b_3, a_4 \vdash b_5, a_6 \vdash b_7 \ldots b_{n-2}, a_{n-1} \vdash b_n$, where each a_i belongs to G and each b_i does not. Since the RDFS rules and triples in the ontology are horn clauses, from a standard result that states that proofs for horn-clause knowledge bases are linear (Nerode and Shore Theorem [16]), the following holds: for a graph G, and a triple pattern t, we have that $G \models_{\texttt{rule}} t$ if and only if there is a linear derivation from G to t.

Now assume that $t_a \prec t_b$. Then there should exist a linear derivation from $\mathrm{cl}(O) \cup \{t_a\}$ to t_b. If the derivation has more than one rule instantiation, because it is linear, we can easily prove that t_b is an indirect relaxation of t_a, a contradiction. □

Proposition 4. Let O be an ontology, o be a derivable triple in $\mathrm{cl}(O)$ and t, t' be triple patterns such that $t, o \vdash t'$. Then t' is an indirect relaxation of t (defined using O).

Proof of Proposition 4. Let $t, o \vdash_s t'$, where s is some rule. We denote by δ this rule instantiation. Now, since o is a derivable triple, there are triples o_1, o_2 such that $o_1, o_2 \vdash_r o$. Because the predicates of the triples in the ontology should be in the set $\{\texttt{type}, \texttt{dom}, \texttt{range}, \texttt{sp}, \texttt{sc}\}$, $r \in \{1, 3, 4\}$. We will prove that there is a triple pattern t'' such that $t, o_1 \vdash t''$ and $t'', o_2 \vdash t'$, and hence t' is an indirect relaxation of t.

We continue the proof for each of the three cases.

- Case $r = 1$. Then without loss of generality $o_1 = (a, \texttt{sp}, b)$, $o_2 = (b, \texttt{sp}, c)$, and $o = (a, \texttt{sp}, c)$. Moreover, $s = 1$ or $s = 2$. If $s = 1$, w.l.g, δ is $(d, \texttt{sp}, a), (a, \texttt{sp}, c) \vdash (d, \texttt{sp}, c)$. That is, $t = (d, \texttt{sp}, a)$ and $t' = (d, \texttt{sp}, c)$. Hence, we have: $(d, \texttt{sp}, a)$, $(a, \texttt{sp}, b) \vdash_1 (d, \texttt{sp}, b)$, $(b, \texttt{sp}, c) \vdash_1 (d, \texttt{sp}, c)$. Therefore, t'' is $(d, \texttt{sp}, b)$.
- Case $r = 3$. Then without loss of generality $o_1 = (a, \texttt{sc}, b)$, $o_2 = (b, \texttt{sc}, c)$, and $o = (a, \texttt{sc}, c)$. Moreover, $s = 3$ or $s = 4$. If $s = 4$, without loss of generality δ is $(x, \texttt{type}, a), (a, \texttt{sc}, c) \vdash_4 (x, \texttt{type}, c)$. That is, $t = (x, \texttt{type}, a)$ and $t' = (x, \texttt{type}, c)$. Hence, we have that: $(x, \texttt{type}, a), (a, \texttt{sc}, b) \vdash_4 (x, \texttt{type}, b), (b, \texttt{sc}, c) \vdash_4 (x, \texttt{type}, c)$. Therefore, $t'' = (x, \texttt{type}, b)$.
- Case $r = 4$. Then without loss of generality $o_1 = (a, \texttt{sc}, b)$, $o_2 = (x, \texttt{type}, a)$, and $o = (x, \texttt{type}, b)$. Moreover, $s = 4$ and without loss of generality δ is $(b, \texttt{sc}, c), (x, \texttt{type}, b) \vdash_4 (x, \texttt{type}, c)$. That is $t = (b, \texttt{sc}, c)$ and $t' = (x, \texttt{type}, c)$. Hence, we have the following linear derivation from t to t': $(b, \texttt{sc}, c), (a, \texttt{sc}, b) \vdash_3 (a, \texttt{sc}, c), (x, \texttt{type}, a) \vdash_4 (x, \texttt{type}, c)$. Consequently, $t'' = (a, \texttt{sc}, c)$. □

Proposition 5. Let O be an ontology and t be a triple pattern not in $\mathrm{cl}(O)$. Then all direct relaxations of t (defined using O) are in the set $\texttt{applyRules}(t, \texttt{red}(O))$.

Proof of Proposition 5. Follows directly from Proposition 4. □

Proposition 6. Let O be an ontology, o be a triple in $\texttt{red}(O)$ and t, t' be triple patterns such that $t, o \vdash t'$. Then t' is an indirect relaxation of t (defined using O) iff o can be derived by applying the rules of Figure 7 starting from $\mathrm{cl}(O)$.

Proof of Proposition 6. (If) It is enough to realize that if $o_1, o_2 \vdash_i o$, where i is some rule in Figure 7, then we have that $t, o_1 \vdash t'', o_2 \vdash t'$, and hence t' is an indirect relaxation of t.

(Only If) Assume that t' is an indirect relaxation of t. If this is the case, we will prove that either $o \notin \texttt{red}(O)$, which yields a contradiction, or o can be obtained by applying the rules of Figure 7 to triples of $\mathrm{cl}(O)$. We will do it by cases. Each case represents that $t, o \vdash t'$ is an instance of a rule i.

- $i = 1$. Then without loss of generality $t = (a, \texttt{sp}, b)$, $o = (b, \texttt{sp}, c)$ and $t' = (a, \texttt{sp}, c)$. Then, w.l.g the derivation δ is of the form $(a, \texttt{sp}, b), (b, \texttt{sp}, d_1) \vdash_1 (a, \texttt{sp}, d_1), (d_1, \texttt{sp}, d_2) \vdash_1 (a, \texttt{sp}, d_2) \dots (a, \texttt{sp}, d_n), (d_n, \texttt{sp}, c) \vdash_1 (a, \texttt{sp}, c)$. Therefore, $o \notin \texttt{red}(O)$, a contradiction.
- $i = 2$. Because o is not a plain triple, without loss of generality $o = (a, \texttt{sp}, b)$, $t = (x, a, y)$ and $t' = (x, b, y)$. Then, w.l.g the derivation δ is $(x, a, y), (a, \texttt{sp}, d_1) \vdash_2 (x, d_1, y), (d_1, \texttt{sp}, d_2) \vdash (x, d_2, y) \dots (x, d_n, y), (d_n, \texttt{sp}, b) \vdash_2 (x, b, y)$. Then, $o \notin \texttt{red}(O)$, a contradiction.
- $i = 3$. The proof is similar to the case where $i = 1$.
- $i = 4$. Then there are two cases: (i) $t = (a, \texttt{sc}, b), o = (x, \texttt{type}, a)$ and $t' = (x, \texttt{type}, b)$. Then, w.l.g the derivation δ is $(a, \texttt{sc}, b), (d_1, \texttt{sc}, a) \vdash (d_1, \texttt{sc}, b), (d_2, \texttt{sc}, d_1) \vdash (d_2, \texttt{sc}, b), \dots \vdash (d_n, \texttt{sc}, b), (x, \texttt{type}, d_n) \vdash (x, \texttt{type}, b)$. Hence, $(d_n, \texttt{sc}, a), (x, \texttt{type}, d_n) \vdash (x, \texttt{type}, a)$, contradicting that $o \notin \texttt{red}(O)$. (ii) $o = (a, \texttt{sc}, b), t = (x, \texttt{type}, a)$ and $t' = (x, \texttt{type}, b)$. Then w.l.g the derivation δ is $(x, \texttt{type}, a), (a, \texttt{sc}, d_1) \vdash (x, \texttt{type}, d_1) \dots (x, \texttt{type}, d_n), (d_n, \texttt{sc}, b) \vdash (x, \texttt{type}, b)$, contradicting that $o \in \texttt{red}(O)$.
- $i = 5$. Then without loss of generality $t = (x, a, y)$, $o = (a, \texttt{dom}, d)$ and $t' = (x, \texttt{type}, d)$. Then, w.l.g the derivation δ is of the form $(x, a, y), (a, \texttt{sp}, b_1) \vdash (x, b_1, y), (b_1, \texttt{sp}, b_2) \vdash_2 \dots (x, b_n, y), (b_n, \texttt{sp}, b) \vdash_2 (x, b, y), (b, \texttt{dom}, c) \vdash_5 (x, \texttt{type}, c), (c, \texttt{sc}, d_1) \vdash_4 (x, \texttt{type}, d_1)(d_1, \texttt{sc}, d_2) \vdash_4 \dots (x, \texttt{type}, d_m), (d_m, \texttt{sc}, d) \vdash_4 (x, \texttt{type}, d)$. Then, we have the following triples in $\mathrm{cl}(O)$: $(a, \texttt{sp}, b)$, $(b, \texttt{dom}, c)$, and $(c, \texttt{sc}, d)$. Then, by rules e1 and e3, $(a, \texttt{dom}, c)$ and $(a, \texttt{dom}, d) = o$ can be derived.
- $i = 6$. The proof is similar to the case where $i = 5$, but now we use rules e2 and e4. □

Proposition 7. Let O be an ontology and t be a triple pattern not in $\mathrm{cl}(O)$. Then $\texttt{applyRules}(t, \texttt{extRed}(O))$ is equal to the set of direct relaxations of t (defined using O).

Proof of Proposition 7. Follows directly from propositions 4 and 6. □

Proposition 8. Let t be a triple pattern and O be an ontology. The relaxation graph of t (using the ontology O) has $O(|\texttt{red}(O)|^2)$ triples.

Proof of Proposition 8. We denote by M_α, where $\alpha \in \{\texttt{type}, \texttt{dom}, \texttt{range}, \texttt{sp}, \texttt{sc}\}$, the number of α-triples in $\texttt{red}(O)$. We prove the proposition by cases.

- t is a `type`-triple. Let $t = (x, \texttt{type}, a)$. Notice that all derivations from t yield triples of the form $(x, \texttt{type}, b)$, for some b mentioned in a `sc`-triple. This is because derivations use only rule 4. Therefore, the relaxation graph of t cannot have more than $M_{\texttt{sc}}$ nodes.
- t is a `dom`-triple or a `range`-triple there are no derivations from t.
- t is a `sc` triple. The relaxation graph of t may only have `sc`-triples and `type`-triples. Assume it only has `sc`-triples. Then, triples cannot be more than the number of pairs of classes in $\texttt{red}(O)$, which is at most $M^2_{\texttt{sc}}$. Now, if the relaxation graph has also `type`-triples, then for each `type`-triple in $\text{cl}(O)$ there are at most $M_{\texttt{sc}}$ `type`-triples in the relaxation graph. So overall we have at most $M^2_{\texttt{sc}} + M_{\texttt{type}} M_{\texttt{sc}}$ triples in the relaxation graph.
- t is a `sp` triple. By a similar argument as in the previous case we have that there are at most $M^2_{\texttt{sp}}$ triples in the relaxation graph.
- t is a plain triple. In this case we have at most $M_{\texttt{sp}} + M_{\texttt{sp}} M_{\texttt{sc}}$ triples in the relaxation graph.

For each triple pattern t' resulting from an ontology relaxation there are at most a constant number of simple relaxations above t', so the generation of simple relaxation does not asymptotically increase the size of the relaxation graph. □

Proposition 9. Let t be a triple pattern and O be an ontology. (i) Computing the direct relaxations of t takes $O(|\texttt{red}(O)|)$ steps. (ii) Computing the relaxation graph of t takes $O(|\texttt{red}(O)|^3)$ steps.

Proof of Proposition 9. Part (i) This is a bound for the cost of evaluating $\texttt{applyRules}(t, \texttt{extRed}(O))$. Part (ii) follows directly from Part (i) and Proposition 8. □

Proposition 10. Let Q be a query and G be a RDF graph. For each query Q' in the relaxation graph of Q, (i) $\texttt{ans}(Q', G) = \bigcup_{Q_i : Q_i \leq_Q Q'} \Delta_{Q_i}(G)$, and (ii) $\texttt{newAnswer}(Q', G) = \Delta_{Q'}(G)$.

Proof of Proposition 10. For simplicity, we consider that Q and Q' have two triple patterns each, and both triple patterns of Q are within the `RELAX` clause. That is $\texttt{Body}(Q) = \{t_1, t_2\}$ and $\texttt{Body}(Q') = \{t'_1, t'_2\}$. The generalization of the proof for more than two triple pattern is direct. Also, we have that $\texttt{Head}(Q) = \texttt{Head}(Q') = H$.

For a triple pattern t' in the relaxation graph of t, let $S_t(t')$ be the set containing t_i such that $t_i \leq_t t'$.

(i) We have that $\texttt{ans}(Q', G) = V_{Q'} = \pi_H(\texttt{find}(t'_1) \bowtie \texttt{find}(t'_2))$.

By the definition of `deltaFind`, it can be easily verified that

$$\texttt{find}(t'_1) = \bigcup_{t_i \in S_t(t'_1)} \texttt{deltaFind}(t_i).$$

A similar equality can be obtained for $\texttt{find}(t'_2)$.

Therefore, we have

$$V_{Q'} = \pi_H((\bigcup_{t_i \in S_t(t'_1)} \texttt{deltaFind}(t_i)) \bowtie (\bigcup_{t_j \in S_t(t'_2)} \texttt{deltaFind}(t_j))),$$

which is equivalent to

$$V_{Q'} = \pi_H((\bigcup_{t_i \in S_t(t'_1), t_j \in S_t(t'_2))} (\texttt{deltaFind}(t_i) \bowtie \texttt{deltaFind}(t_j))).$$

which is equivalent to

$$V_{Q'} = \bigcup_{Q_i : Q_i \leq_Q Q'} \Delta_{Q_i}(G).$$

(ii) We use (i) to replace $\texttt{newAnswer}(Q', G)$, obtaining

$$\Delta_{Q'}(G) = (\bigcup_{Q_i : Q_i \leq_Q Q'} \Delta_{Q_i}(G)) - (\bigcup_{Q_i : Q_i \leq_Q Q', Q_i \neq Q'} \texttt{ans}(Q_i, G))$$

But, from (i) it follows that:

$$\bigcup_{Q_i : Q_i \leq_Q Q'} \Delta_{Q_i}(G)) = \bigcup_{Q_i : Q_i \leq_Q Q'} \texttt{ans}(Q_i, G)).$$

Hence, (ii) is equivalent to $\Delta_{Q'}(G) = \Delta_{Q'}(G)$. □

Proposition 11. The algorithm `RelaxEval` returns its tuples in ranked order.

Proof of Proposition 11. Follows directly from Proposition 10 and the fact that `RelaxEval` traverses the relaxation graph of Q in breadth-first fashion, that is, delta views of less relaxed queries are processed before delta views of more relaxed queries. □

Proposition 12. Let Q be a query, O be an ontology and G an RDF graph. Then $\texttt{RelaxEval}(Q, G, k)$ runs in time $O(m^{2n}|G|^n)$, where m is the number of triples in $\texttt{red}(O)$, and $n = |\texttt{Body}(Q)|$.

Proof of Proposition 12. Follows from the fact that the size of the relaxation graph is in $O(m^{2n})$ and the execution of each delta view takes time in $O(|G|^n)$.□

Proposition 13. The simple relaxation relation $\leq_{\texttt{simple}}$ is a partial order up to triple pattern isomorphism.

Proof of Proposition 13. It can be easily verified that $\leq_{\texttt{simple}}$ is reflexive and transitive, and it is also reflexive and transitive up to isomorphism. So we have to prove that it is antisymmetric up to isomorphism. If it is not the case, there are two triples t_a, t_b, where t_a is not isomorphic to t_b, such that $t_a \leq_{\texttt{simple}} t_b$ and $t_b \leq_{\texttt{simple}} t_a$. Hence there are maps u_a from t_a to t_b and u_b from t_b to t_a and both maps preserve the fixed variables. Therefore, t_a is isomorphic to t_b, a contradiction. □