Reformulation-Based Query Answering for RDF Graphs with RDFS Ontologies

Maxime Buron[1,2](✉), François Goasdoué[3](✉), Ioana Manolescu[1,2](✉), and Marie-Laure Mugnier[4](✉)

[1] Inria Saclay, Palaiseau, France
{maxime.buron,ioana.manolescu}@inria.fr
[2] LIX (UMR 7161, CNRS and Ecole polytechnique), Palaiseau, France
[3] Univ Rennes, CNRS, IRISA, Lannion, France
fg@irisa.fr
[4] Univ. Montpellier, LIRMM, Inria, Montpellier, France
mugnier@lirmm.fr

Abstract. Query answering in RDF knowledge bases has traditionally been performed either through graph saturation, i.e., adding all implicit triples to the graph, or through query reformulation, i.e., modifying the query to look for the explicit triples entailing precisely what the original query asks for. The most expressive fragment of RDF for which Reformulation-based query answering exists is the so-called database fragment [13], in which implicit triples are restricted to those entailed using an RDFS ontology. Within this fragment, query answering was so far limited to the interrogation of data triples (non-RDFS ones); however, a powerful feature specific to RDF is the ability to query data and schema triples together. In this paper, we address the general query answering problem by reducing it, through a pre-query reformulation step, to that solved by the query reformulation technique of [13]. We also report on experiments demonstrating the low cost of our reformulation algorithm.

Keywords: Query answering · Query reformulation · RDF · RDFS

1 Introduction

RDF is the standard model for sharing data and knowledge bases. The rapid increase in number and size of RDF graphs makes efficient query answering on RDF quite a challenging task. *Reasoning* raises a performance challenge: query answering on an RDF graph no longer reduces to *evaluating* the query on the graph (by finding all the homomorphisms, or embeddings, of the query in the graph). Instead, it requires taking into account also the possible ontology (or knowledge) rules, which specify how different classes and properties of an RDF graph relate to each other, and may lead to query answers that evaluation alone cannot compute. Moreover, SPARQL, the standard query language of RDF, allows *querying the data and the ontology together.* This is a radical departure both from relational databases, and from Description Logics (DL)-style models for RDF data and queries.

For what concerns reasoning, two main methods have been explored: graph saturation, which injects the ontology knowledge into the graph, and query reformulation, which pushes it into the query. Saturation adds to the graph all the triples it entails through the ontology. Evaluating a query on a saturated graph can be quite efficient; however, saturation takes time to compute, space to store, and needs to be updated when the data and/or ontology rules change. Reformulation leaves the graph unchanged and builds a reformulated query which, evaluated on the original graph, computes all the answers, including those that hold due to entailed triples. Each query reformulation method, thus, targets a certain ontology language and a query dialect. The most expressive RDF fragment for which sound and complete reformulation-based query answering exists is the so-called database fragment [13], in which RDF Schema (RDFS, in short) is used to describe the ontology, while queries only carry over the data triples.

In this work, we present a novel *reformulation-based query answering* under *RDFS* ontologies for *Basic Graph Pattern (BGP) queries over both the data and the ontology.* This goes beyond the closest algorithm previously known [13] which is restricted to queries over the data only (not over the ontology). The algorithm we present here also goes beyond those of RDF platforms such as Jena, Virtuoso or Stardog, which we found experimentally to be incomplete when answering through reformulation queries over the data and the ontology of an RDF graph. Below, we recall some terminology (Sect. 2) and discuss the state of the art (Sect. 3). Then, Sect. 4 introduces our novel query reformulation algorithm, which we implemented in the platform used in [10,13], leveraging an efficient relational database (RDBMS) engine for query answering. Our experiments (Sect. 5) demonstrate the practical interest of our reformulation approach. Detailed proofs are available in [9].

2 Preliminaries

We present the basics of the RDF graph data model (Sect. 2.1), of RDF entailment used to make explicit the implicit information RDF graphs encode (Sect. 2.2), as well as how they can be queried using the widely-considered SPARQL Basic Graph Pattern queries (Sect. 2.3).

2.1 RDF Graph

We consider three pairwise disjoint sets of values: $\mathscr{I}$ of IRIs (resource identifiers), $\mathscr{L}$ of literals (constants) and $\mathscr{B}$ of blank nodes modeling unknown IRIs or literals, a.k.a. to *labelled nulls* [4]. A *well-formed triple* belongs to $(\mathscr{I} \cup \mathscr{B}) \times \mathscr{I} \times (\mathscr{L} \cup \mathscr{I} \cup \mathscr{B})$, and an *RDF graph* G is a set of well-formed triples. A triple $(\mathtt{s}, \mathtt{p}, \mathtt{o})$ states that its *subject* $\mathtt{s}$ has the *property* $\mathtt{p}$ with the *object* value $\mathtt{o}$ [1]. We denote by $\mathrm{Val}(G)$ the set of all values (IRIs, blank nodes and literals) occurring in an RDF graph G, and by $\mathrm{Bl}(G)$ its set of blank nodes.

Table 1. RDF statements.

RDF assertions	Triple notation
Class assertion	$(\mathtt{s}, \tau, \mathtt{o})$
Property assertion	$(\mathtt{s}, \mathtt{p}, \mathtt{o})$ with $\mathtt{p} \notin \{\tau, \prec_{sc}, \prec_{sp}, \hookleftarrow_d, \hookrightarrow_r\}$
RDFS constraints	Triple notation
Subclass	$(\mathtt{s}, \prec_{sc}, \mathtt{o})$
Subproperty	$(\mathtt{s}, \prec_{sp}, \mathtt{o})$
Domain typing	$(\mathtt{s}, \hookleftarrow_d, \mathtt{o})$
Range typing	$(\mathtt{s}, \hookrightarrow_r, \mathtt{o})$

Within an RDF graph, triples model either factual *assertions* for unary relations called *classes* and binary relations called *properties*, or *RDFS ontological constraints* between classes and properties. The RDFS constraints are of four flavours: **subclass** constraints, **subproperty** constraints, typing of the **domain** (first attribute) or of the **range** (second attribute) of a property. The triple notations we adopt for RDF assertions and constraints are shown in Table 1. In a triple, we use $\_{:}b$ (possibly with indices) to denote blank nodes, and strings between quotes to denote literals.

We consider RDF graphs with *RDFS ontologies*, i.e., constraints of the four flavors above, excluding constraints which would alter the commonly-accepted RDFS semantics. For instance, $(\hookleftarrow_d, \prec_{sp}, \hookrightarrow_r)$ is not allowed as it would make domain typing a particular case of range typing. Let RDFS(G) denote the RDFS constraints of a graph G. We define:

Definition 1 (RDF graph with an RDFS ontology). *An* RDFS ontology *(or ontology in short) is a set of RDFS constraints, whose subjects and objects are either IRIs (other than $\prec_{sc}, \prec_{sp}, \hookleftarrow_d, \hookrightarrow_r$, τ) or blank nodes.*

An RDF *graph G with ontology O is such that:* RDFS$(G) = O$.

Example 1 (Running example). Consider the following RDF graph:

$$\begin{aligned} G_{\text{ex}} = \{&(\text{:worksFor}, \hookleftarrow_d, \text{:Person}), (\text{:worksFor}, \hookrightarrow_r, \text{:Org}), (\text{:PubAdmin}, \prec_{sc}, \text{:Org}) \\ &(\text{:Comp}, \prec_{sc}, \text{:Org}), (\_{:}b_C, \prec_{sc}, \text{:Comp}), (\text{:hiredBy}, \prec_{sp}, \text{:worksFor}) \\ &(\text{:ceoOf}, \prec_{sp}, \text{:worksFor}), (\text{:ceoOf}, \hookrightarrow_r, \text{:Comp}), \\ &(\text{:p}_1, \text{:ceoOf}, \text{:c}), (\text{:c}, \tau, \_{:}b_C), (\text{:p}_2, \text{:hiredBy}, \text{:a}), (\text{:a}, \tau, \text{:PubAdmin})\} \end{aligned}$$

The ontology of G_{ex}, i.e., the first eight triples, states that persons are working for organizations, some of which are public administrations or companies. Further, there exists a special kind of company (modeled by $\_{:}b_C$). Being hired by or being CEO of an organization are two ways of working for it; in the latter case, this organization is a company. The assertions of G_{ex}, i.e., the four remaining triples, states that :p$_1$ is CEO of :c, which is a company of the special kind $\_{:}b_C$, and :p$_2$ is hired by the public administration :a.

Table 2. RDFS entailment rules.

Rule [2]	Entailment rule
`rdfs2`	$(\mathtt{p}, \hookleftarrow_d, \mathtt{o}), (\mathtt{s}_1, \mathtt{p}, \mathtt{o}_1) \rightarrow (\mathtt{s}_1, \tau, \mathtt{o})$
`rdfs3`	$(\mathtt{p}, \hookrightarrow_r, \mathtt{o}), (\mathtt{s}_1, \mathtt{p}, \mathtt{o}_1) \rightarrow (\mathtt{o}_1, \tau, \mathtt{o})$
`rdfs5`	$(\mathtt{p}_1, \prec_{sp}, \mathtt{p}_2), (\mathtt{p}_2, \prec_{sp}, \mathtt{p}_3) \rightarrow (\mathtt{p}_1, \prec_{sp}, \mathtt{p}_3)$
`rdfs7`	$(\mathtt{p}_1, \prec_{sp}, \mathtt{p}_2), (\mathtt{s}, \mathtt{p}_1, \mathtt{o}) \rightarrow (\mathtt{s}, \mathtt{p}_2, \mathtt{o})$
`rdfs9`	$(\mathtt{s}, \prec_{sc}, \mathtt{o}), (\mathtt{s}_1, \tau, \mathtt{s}) \rightarrow (\mathtt{s}_1, \tau, \mathtt{o})$
`rdfs11`	$(\mathtt{s}, \prec_{sc}, \mathtt{o}), (\mathtt{o}, \prec_{sc}, \mathtt{o}_1) \rightarrow (\mathtt{s}, \prec_{sc}, \mathtt{o}_1)$
`ext1`	$(\mathtt{p}, \hookleftarrow_d, \mathtt{o}), (\mathtt{o}, \prec_{sc}, \mathtt{o}_1) \rightarrow (\mathtt{p}, \hookleftarrow_d, \mathtt{o}_1)$
`ext2`	$(\mathtt{p}, \hookrightarrow_r, \mathtt{o}), (\mathtt{o}, \prec_{sc}, \mathtt{o}_1) \rightarrow (\mathtt{p}, \hookrightarrow_r, \mathtt{o}_1)$
`ext3`	$(\mathtt{p}, \prec_{sp}, \mathtt{p}_1), (\mathtt{p}_1, \hookleftarrow_d, \mathtt{o}) \rightarrow (\mathtt{p}, \hookleftarrow_d, \mathtt{o})$
`ext4`	$(\mathtt{p}, \prec_{sp}, \mathtt{p}_1), (\mathtt{p}_1, \hookrightarrow_r, \mathtt{o}) \rightarrow (\mathtt{p}, \hookrightarrow_r, \mathtt{o})$

A *homomorphism between RDF graphs* allows characterizing whether an RDF graph *simply entails* another, based on their explicit triples only:

Definition 2 (RDF graph homomorphism). *Let G and G' be two RDF graphs. A* homomorphism *from G to G' is a function φ from* $\mathrm{Val}(G)$ *to* $\mathrm{Val}(G')$, *which is the identity on IRIs and literals, such that for any triple (s, p, o) in G, the triple $(\varphi(s), \varphi(p), \varphi(o))$ is in G'.*

Note that, according to the previous definition, a G blank node can be mapped to any G' value. A graph G' simply entails a graph G if there is a homomorphism φ from G to G', which we denote by $G' \models^{\varphi} G$.

2.2 RDF Entailment Rules

The semantics of an RDF graph consists of the explicit triples it contains, and of the implicit triples that can be derived from it using *RDF entailment rules.*

Definition 3 (RDF entailment rule). *An* RDF entailment rule *r has the form body$(r) \rightarrow$ head(r), where body(r) and head(r) are RDF graphs, respectively called* body *and* head *of the rule r.*

The standard RDF entailment rules are defined in [2]. In this work, we consider the rules shown in Table 2, which we call *RDFS entailment rules*; all values except the $\tau, \prec_{sc}, \prec_{sp}, \hookleftarrow_d, \hookrightarrow_r$ properties are blank nodes. These rules are the most frequently used for RDFS entailment; they produce implicit triples by exploiting the RDFS ontological constraints of an RDF graph. For example, the rule `rdfs9`, which propagates values from subclasses to their superclasses, is defined by body(`rdfs9`) $= \{(\mathtt{s}, \prec_{sc}, \mathtt{o}), (\mathtt{s}_1, \tau, \mathtt{s})\}$ and head(`rdfs9`) $= \{(\mathtt{s}_1, \tau, \mathtt{o})\}$. The *direct entailment* of an RDF graph G with a set of RDF entailment rules

$\mathcal{R}$, denoted by $C_{G,\mathcal{R}}$, characterizes the set of implicit triples resulting from rule applications that use solely the explicit triples of G. It is defined as:

$$C_{G,\mathcal{R}} = \{\varphi(\text{head}(r)) \mid r \in \mathcal{R}, G \models^{\varphi} \text{body}(r)\}$$

For instance, the rule `rdfs9` applies to the graph G_{ex}: $G_{\text{ex}} \models^{\varphi} \text{body}(\texttt{rdfs9})$ through the homomorphism φ defined as $\{\texttt{s} \mapsto \_{:}b_C, \texttt{o} \mapsto \text{:Comp}, \texttt{s}_1 \mapsto \text{:c}\}$, hence allows deriving the implicit triple $(\text{:c}, \tau, \text{:Comp})$.

The *saturation* of an RDF graph allows materializing its semantics, by iteratively augmenting it with the triples it entails using a set $\mathcal{R}$ of RDF entailment rules, until reaching a fixpoint; this process is finite [2]. Formally:

Definition 4 (RDF graph saturation). *Let G be an RDF graph and $\mathcal{R}$ a set of entailment rules. We recursively define a sequence $(G_i^{\mathcal{R}})_{i\in\mathbb{N}}$ of RDF graphs as follows: $G_0^{\mathcal{R}} = G$ and $G_{i+1}^{\mathcal{R}} = G_i^{\mathcal{R}} \cup C_{G_i^{\mathcal{R}},\mathcal{R}}$ for $0 \leq i$. The* saturation *of G w.r.t. $\mathcal{R}$, denoted by $G^{\mathcal{R}}$, is $G_n^{\mathcal{R}}$ for n the smallest integer such that $G_n^{\mathcal{R}} = G_{n+1}^{\mathcal{R}}$.*

Example 2. The saturation of G_{ex} w.r.t. the set $\mathcal{R}$ of RDFS entailment rules shown in Table 2 is attained after the following *two* saturation steps:

$$\begin{aligned}(G_{\text{ex}})_1^{\mathcal{R}} =& G_{\text{ex}} \cup \{(\_{:}b_C, \prec_{sc}, \text{:Org}), (\text{:hiredBy}, \hookleftarrow_d, \text{:Person}), (\text{:hiredBy}, \hookrightarrow_r, \text{:Org}),\\ & (\text{:ceoOf}, \hookleftarrow_d, \text{:Person}), (\text{:ceoOf}, \hookrightarrow_r, \text{:Org}),\\ & (\text{:p}_1, \text{:worksFor}, \text{:c}), (\text{:c}, \tau, \text{:Comp}), (\text{:p}_2, \text{:worksFor}, \text{:a}), (\text{:a}, \tau, \text{:Org})\}\\ (G_{\text{ex}})_2^{\mathcal{R}} =& (G_{\text{ex}})_1^{\mathcal{R}} \cup \{(\text{:p}_1, \tau, \text{:Person}), (\text{:p}_2, \tau, \text{:Person}), (\text{:c}, \tau, \text{:Org})\}\end{aligned}$$

Simple entailment between RDF graphs, which is based on their explicit triples only, generalizes to *entailment between RDF graphs w.r.t. a set of RDF entailment rules*, to also take into account their implicit triples.

A graph G entails a graph G' w.r.t. a set of rules $\mathcal{R}$, noted $G \models_{\mathcal{R}}^{\varphi} G'$, whenever there is a homomorphism φ from G' to $G^{\mathcal{R}}$. Of course, simple entailment and entailment between RDF graphs coincide when $\mathcal{R} = \emptyset$. For simplicity, we will just write $G \models_{\mathcal{R}} G'$ whenever φ is not needed for the discussion.

In this work, unless otherwise specified, $\mathcal{R}$ denotes the rules from Table 2.

2.3 Basic Graph Pattern Queries

A popular fragment of the SPARQL query language consists of conjunctive queries, also known as basic graph pattern queries. Let $\mathscr{V}$ be a set of variable symbols, disjoint from $\mathscr{I} \cup \mathscr{B} \cup \mathscr{L}$. A *basic graph pattern* (BGP) is a set of *triple patterns* (triples in short) belonging to $(\mathscr{I} \cup \mathscr{B} \cup \mathscr{V}) \times (\mathscr{I} \cup \mathscr{V}) \times (\mathscr{I} \cup \mathscr{B} \cup \mathscr{L} \cup \mathscr{V})$. For a BGP P, we denote by Var(P) the set of variables occurring in P, by Bl(P) its set of blank nodes, and by Val(P) its set of values (IRIs, blank nodes, literals and variables).

Definition 5 (BGP query). *A* BGP query *(BGPQ) q is of the form $q(\bar{x}) \leftarrow P$, where P is a BGP also denoted by body(q) and $\bar{x} \subseteq$* Var(P) *is the set of q's answer variables. The* arity *of q is that of $\bar{x}$, i.e., $|\bar{x}|$.*

Partially instantiated BGPQs generalize BGPQs and have been used for reformulation-based query answering [13]. Starting from a BGPQ q, partial instantiation replaces *some* variables and/or blank nodes with values from $\mathscr{I} \cup \mathscr{L} \cup \mathscr{B}$, as specified by a substitution σ; the partially instantiated query is denoted q_σ. Observe that when $\sigma = \emptyset$, q_σ coincides with q. Further, due to σ, and in contrast with standard BGPQs, some answer variables of q_σ can be bound:

Example 3. Consider the BGPQ asking for *who is working for which kind of company*: $q(x, y) \leftarrow (x, \text{:worksFor}, z), (z, \tau, y), (y, \prec_{sc}, \text{:Comp})$, and the substitution $\sigma = \{x \mapsto \text{:p}_1\}$. The partially instantiated BGPQ q_σ corresponds to $q(\text{:p}_1, y) \leftarrow (\text{:p}_1, \text{:worksFor}, z), (z, \tau, y), (y, \prec_{sc}, \text{:Comp})$.

The semantics of a (partially instantiated) BGPQ on an RDF graph is defined through homomorphisms from the query body to the saturation of the queried graph. The homomorphisms needed here are a straightforward extension of RDF graph homomorphisms (Definition 2) to also take variables into account.

Definition 6 ((Non-standard) BGP to RDF graph homomorphism). *A* homomorphism *from a BGP q to an RDF graph G is a function φ from* $\mathrm{Val}(body(q))$ *to* $\mathrm{Val}(G)$ *such that for any triple $(s, p, o) \in body(q)$, the triple $(\varphi(s), \varphi(p), \varphi(o))$ is in G. For a* standard *homomorphism, as per the SPARQL recommendation, φ is the identity on IRIs and literals; for a non* non-standard *one, φ is the identity on IRIs, literals <u>and</u> on blank nodes.*

We distinguish *query evaluation*, whose result is just based on the explicit triples of the graph, i.e., on BGP to RDF graph homomorphisms, from *query answering* that also accounts for the implicit graph triples, i.e., based on both BGP to RDF graph homomorphisms *and* RDF entailment. In this paper, we use two flavors of query evaluation and of query answering, which differ in relying either on standard or on non-standard BGP to RDF graph homomorphisms.

Definition 7 ((Non-standard) evaluation and answering). *Let q_σ be a partially instantiated BGPQ q_σ obtained from a BGPQ q and a substitution σ. The* standard answer set *to q_σ on an RDF graph G w.r.t. a set $\mathcal{R}$ of RDF entailment rules is: $q_\sigma(G, \mathcal{R}) = \{\varphi(\bar{x}_\sigma) \mid G \models^{\varphi}_{\mathcal{R}} body(q)_\sigma\}$ where $\bar{x}_\sigma$ and $body(q)_\sigma$ denote the result of replacing the variables and blank nodes in $\bar{x}$ and $body(q)$, respectively, according to σ.*

If $\bar{x} = \emptyset$, q_σ is a Boolean *query, in which case q_σ is false when $q_\sigma(G, \mathcal{R}) = \emptyset$ and true when $q_\sigma(G, \mathcal{R}) = \{\langle\rangle\}$, i.e., the answer to q_σ is an empty tuple.*

We call $q_\sigma(G, \emptyset)$ the standard evaluation *of q_σ on G, written $q_\sigma(G)$ for short, which solely amounts to standard BGP to RDF graph homomorphism finding.*

The non-standard answer set, *denoted $\overbrace{q_\sigma(G, \mathcal{R})}$, and* non-standard evaluation *$\overbrace{q_\sigma(G)}$ of q_σ on G w.r.t. $\mathcal{R}$ only differ from the standard ones by using non-standard BGP to RDF graph homomorphisms.*

These notions and notations naturally extend to *unions* of BGPQs.

Example 4. Consider again the BGPQs from the preceding example. Their standard evaluations on G_{ex} are empty because G_{ex} has no explicit :worksFor assertion, while their standard answer sets on G_{ex} w.r.t. $\mathcal{R}$ are $\{\langle \text{:p}_1, \_\text{:}b_C \rangle\}$ because :p$_1$ being CEO of :c, :p$_1$ implicitly works for it, and :c is explicitly a company of the particular unknown type $\_\text{:}b_C$.

Consider now the BGPQ $q(x) \leftarrow (x, \text{:worksFor}, y), (y, \tau, \_\text{:}b_C)$. Under standard query answering, it asks for *who is working for some kind of organization* and its answer set is $\{\langle \text{:p}_1 \rangle, \langle \text{:p}_2 \rangle\}$; by contrast, under non-standard query answering, it asks for *who is working for an organization of the particular unknown type* $\_\text{:}b_C$ *in* G_{ex} and its answer set is just $\{\langle \text{:p}_1 \rangle\}$.

3 Prior Related Work

Two main techniques for answering BGPQs on RDF graphs have been investigated in the literature.

Saturation-Based Query Answering. This technique directly follows from the definition of query answers in the W3C's SPARQL recommendations [3], recalled in Sect. 2.3 for BGPQs. Indeed, it trivially follows from Definition 7 that $q(G, \mathcal{R}) = q(G^{\mathcal{R}})$ (resp. $\overbrace{q(G, \mathcal{R})} = \overbrace{q(G^{\mathcal{R}})}$), i.e., query answering reduces to query evaluation on the *saturated* RDF graph. Saturation-based query answering is typically fast, because it only requires query evaluation, which can be efficiently performed by a data management engine. However, saturation takes time to be computed, requires extra space to be stored, and must be recomputed or maintained (e.g., [7,8,13]) upon updates. Many RDF data management systems use saturation-based query answering. They either allow computing graph saturation, e.g., Jena and RDFox, or simply assume that RDF graphs have been saturated before being stored, e.g., DB2RDF.

Reformulation-Based Query Answering. This technique also reduces query answering to query evaluation, however, the reasoning needed to ensure complete answers is performed on the query and not on the RDF graph. A given query q, asked on an RDF graph G w.r.t. $\mathcal{R}$ is *reformulated* into a query q' such that $q(G, \mathcal{R}) = q'(G)$ or $q(G, \mathcal{R}) = \overbrace{q'(G)}$ holds. Standard or non-standard query evaluation is needed on the reformulated query, depending on the considered RDF fragment: when blank nodes are allowed in RDFS constraints, non-standard evaluation is used [13], while standard evaluation is sufficient otherwise [5,12]. Different SPARQL dialects have been adopted for BGPQ reformulation in more limited settings than the one considered in this paper, i.e., the database fragment of RDF and unrestricted BGPQs. *Unions of BGPQs (UBGPQs in short)* have been used in [5,12,13]. However, these works are restricted to input BGPQs that must be matched on *RDF assertions* only. BGPQs aiming at interrogating solely the RDFS ontology, or the ontology *and* the assertions are not considered, even though such joint querying is a major novelty of RDF and SPARQL. The techniques adopt unions of BGPQs [5] or of *partially instantiated* BGPQs [12,13], depending on whether variables can be used in class and property positions in queries, e.g., whether a query triple (x, τ, z) or (x, y, z) is allowed. Reformulation-based query answering in the DL fragment of RDF, which is strictly contained in the database fragment of RDF, has been investigated for relational conjunctive queries [5,11], while the slight extension thereof considered in [6,12,14,18] has been investigated for one-triple BGPQs [14,18], BGPQs [12], and SPARQL queries [6]. In [6], SPARQL queries are reformulated into *nested* SPARQL, allowing nested regular expressions in property position in query triples. These reformulations allow sound and complete query answering on restricted RDF graphs with RDFS ontologies: these graph *must not contain blank nodes*. While such nested reformulations are more compact, the queries we produce are more practical, since their evaluation can be delegated to any off-the-shelf RDBMS, or to an RDF engine such as RDF-3X [17] even if it is unaware of reasoning; further, we do not impose restrictions on RDF graphs.

In Sect. 4, we devise a reformulation-based query answering technique for the entire database fragment of RDF and unrestricted BGPQs.

Reformulation-based query answering is well-suited to frequently updated RDF graphs, because it uses the queried RDF graph at query time (and not its saturation). However, reformulated queries tend to be more complex than the original ones, thus costly to evaluate. To mitigate this, [10] provides an *optimized reformulation framework* whereas an incoming BGPQ is reformulated into a *join of unions of BGPQs (JUBGPQ in short)*. This approach being based on a database-style *cost model*, JUBGPQ reformulations are very efficiently evaluated.

Some available RDF data management systems use reformulation-based query answering but return incomplete answer sets in the RDF setting we consider[1], e.g., AllegroGraph[2] and Stardog[3] miss answers because they cannot evaluate triples with a variable property on the schema, while Virtuoso[4] only exploits subclass and subproperty constraints, but not domain and range ones. Finally, *Hybrid* approaches have also been studied, e.g., in [19], where some one-triple queries are chosen for materialization and reused during reformulation-based answering.

4 Extending Query Reformulation to Queries over the Ontology

We now present the main contribution of this paper: a reformulation-based query answering (QA) technique able to compute all answers to a BGPQ against *all* the explicit and implicit triples of an RDF graph, i.e., its RDF assertions *and* RDFS constraints, as per the SPARQL and RDF recommendations [2,3]. The central idea is to *reduce* this full QA problem to an *assertion-level* QA, i.e., where the query is confined to just the explicit and implicit RDF assertions.

[1] See discussion at https://team.inria.fr/cedar/rdfs-reasoning-experiments/.
[2] https://franz.com/agraph/support/documentation/current/reasoner-tutorial.html.
[3] https://www.stardog.com/docs/#_owl_rule_reasoning.
[4] http://docs.openlinksw.com/virtuoso/rdfsparqlruleimpl.

To this aim, we divide query reformulation in two steps: the first reformulation step implements the reduction, while the second step relies on the reformulation technique of [13], which considers assertion-level QA.

4.1 Overview of Our Query Reformulation Technique

Let us first notice that the body of any BGPQ q can be divided into three disjoint subsets of triples (s, p, o), according to the nature of term p: the set b_c of RDFS triples where p is a built-in RDFS property ($\prec_{sc}$, $\prec_{sp}$, $\hookleftarrow_d$, $\hookrightarrow_r$); the set b_a of assertion triples where p is τ or a user-defined property; and the set b_v where p is a variable. We denote by q_c, q_a and q_v the subqueries respectively associated with these bodies. If b_v is not empty, q can be reformulated as a union of BGPQs, say $\mathcal{Q}$, composed of all BGPQs that can be obtained from q by substituting some (possibly none) variables occurring in q_v with one of the four built-in RDFS properties. We assume this preprocessing step to simplify the explanations, even if in practice it may not be performed. Then, the answers to any BGPQ $q' \in \mathcal{Q}$ can be computed in two steps:

1. compute the answers to the subquery q'_c, i.e., with body restricted to the RDFS triples; if q'_c has no answer, neither has q'. Otherwise, each answer to q'_c defines a (partial) instantiation σ of the variables in q'.
2. compute the assertion-level answers to each partially instantiated query $(q'_{a,v})_\sigma$, where $q'_{a,v}$ is the subquery with body $b'_a \cup b'_v$, and return the union of all the obtained answers.

To summarize, Step 1 computes answers to RDFS triples, which allows one to produce a set of partially instantiated queries that no longer contain RDFS triples. Hence, these queries can then be answered using RDF assertions only, which is the purpose of Step 2. Our two-step query reformulation follows this decomposition. It furthermore considers a partition of the set $\mathcal{R}$ of RDFS entailment rules (recall Table 2) into two subsets: the set of rules $\mathcal{R}_c$ that produces *RDFS constraints* and the set of rules $\mathcal{R}_a$ that produces *RDF assertions*:

- $\mathcal{R}_c = \{\texttt{rdfs5}, \texttt{rdfs11}, \texttt{ext1}, \texttt{ext2}, \texttt{ext3}, \texttt{ext4}\}$;
- $\mathcal{R}_a = \{\texttt{rdfs2}, \texttt{rdfs3}, \texttt{rdfs7}, \texttt{rdfs9}\}$.

The reason of this decomposition is that query answering remains complete if, on the one hand, only $\mathcal{R}_c$ is considered to answer queries made of RDFS triples (Step 1: for any graph G, $q'_c(G, \mathcal{R}) = q'_c(G, \mathcal{R}_c)$), and, on the other hand, only $\mathcal{R}_a$ is considered to answer queries on RDF assertions only, as shown in [13].

Query reformulation does not directly work on the entailment rules as classical backward-chaining techniques would do. Instead, a set of so-called *reformulation rules* is specifically associated with $\mathcal{R}_c$ (resp. $\mathcal{R}_a$). We can now outline the two-step query reformulation algorithm:

Step 1. Reformulation w.r.t. $\mathcal{R}_c$: The input BGPQ q is first reformulated into a union $\mathcal{Q}_c$ of partially instantiated BGPQs, using the set of reformulation rules associated with $\mathcal{R}_c$ (see Fig. 1). This reformulation step is sound and complete for query answering w.r.t. $\mathcal{R}_c$, i.e., for any graph G, $q(G, \mathcal{R}_c) = \overbrace{\mathcal{Q}_c(G)}$; furthermore, it preserves the answers with respect to the set $\mathcal{R}$, i.e., $q(G, \mathcal{R}_c \cup \mathcal{R}_a) = \overbrace{\mathcal{Q}_c(G, \mathcal{R}_c \cup \mathcal{R}_a)}$ (see Theorem 1).

Step 2. Reformulation w.r.t. $\mathcal{R}_a$: We recall that $\mathcal{Q}_c$ consists of queries that do not contain RDFS triples. It is given as input to the query reformulation algorithm of [13], which relies on a set of reformulation rules associated with $\mathcal{R}_a$ to output a union $\mathcal{Q}_{c,a}$ of partially instantiated BGPQs. This reformulation step being sound and complete for query answering on the RDF assertions of an RDF graph, we obtain the soundness and completeness of the two-step reformulation, i.e., $q(G, \mathcal{R}_c \cup \mathcal{R}_a) = \overbrace{\mathcal{Q}_c(G, \mathcal{R}_a)} = \overbrace{\mathcal{Q}_{c,a}(G)}$ (see Theorem 2).

4.2 Reformulation Rules Associated with $\mathcal{R}_c$

We now detail reformulation rules associated with $\mathcal{R}_c$, see Fig. 1. Each reformulation rule is of the form $\frac{input}{output}$, where the input is composed of a triple from a partially instantiated query q_σ and a triple from O and the output is a new query obtained from q_σ by instantiating a variable, removing the input triple, or replacing it by one or two triples. The notation *old triple/new triple(s)* means that *old triple* is replaced by *new triple(s)*. The specific case where *old triple* is simply removed is denoted by *old triple*/−. The notations for the triples themselves are the following:

- a bold character like $\mathtt{c}$, $\mathtt{p}$, $\mathtt{s}$ or $\mathtt{o}$ represents an IRI or a blank node
- a v character represents a variable of the query
- s and o characters represent either variables, IRIs or blank nodes, in subject and object positions respectively.

The four rules (1) substitute a variable in a property position by one of the four built-in RDFS properties. All the other rules take as input query triples of the form $(s, \mathtt{p}, o)$, where $\mathtt{p}$ is a built-in RDFS property. Rule (2) simply removes from q_σ an (instantiated) input triple found in O.

Query triples with a domain ($\hookleftarrow_d$) or range property ($\hookrightarrow_r$) are processed by Rules (3)–(11). Given a triple $(\mathtt{p}, \hookleftarrow, \mathtt{c})$ in O (where $\hookleftarrow$ stands for $\hookleftarrow_d$ or $\hookrightarrow_r$), Rule (3) replaces a query triple of the form $(v_1, \hookleftarrow, v_2)$ by two triples $(v_1, \prec_{sp}, \mathtt{p})$ and $(\mathtt{c}, \prec_{sc}, v_2)$. This rule relies on the fact that a triple $(\mathtt{p}', \hookleftarrow, \mathtt{c}')$ belongs to the saturation of the RDF graph by $\mathcal{R}_c$ if and only if $\mathtt{p}'$ is a subproperty of $\mathtt{p}$ (including $\mathtt{p} = \mathtt{p}'$) and $\mathtt{c}$ is a subclass of $\mathtt{c}'$ (including $\mathtt{c} = \mathtt{c}'$), see Lemma 1 in Sect. 4.3. However, we do not assume that the ontology ensures the reflexivity of the subclass and subproperty relations, hence Rules (4)–(7), whose sole purpose is to deal with the cases $\mathtt{c} = \mathtt{c}'$ and $\mathtt{p} = \mathtt{p}'$. Should the ontology contain axiomatic triples ensuring the reflexivity of subclass and subproperty, these four rules would be useless. Note that a natural candidate rule to deal with the case where $\mathtt{c} \neq \mathtt{c}'$ and $\mathtt{p} \neq \mathtt{p}'$ would have been the following:

$$\frac{(\mathtt{p}', \hookleftarrow, \mathtt{c}') \in q_\sigma, (\mathtt{p}, \hookleftarrow, \mathtt{c}) \in O}{q_{\sigma[(\mathtt{p}', \hookleftarrow, \mathtt{c}')/(\mathtt{p}', \prec_{sp}, \mathtt{p}), (\mathtt{c}, \prec_{sc}, \mathtt{c}')]}} \tag{17}$$

$$\frac{(s,v,o)\in q_\sigma}{q_{\sigma\cup\{v\to\prec_{sc}\}}},\frac{(s,v,o)\in q_\sigma}{q_{\sigma\cup\{v\to\prec_{sp}\}}},\frac{(s,v,o)\in q_\sigma}{q_{\sigma\cup\{v\to\hookleftarrow_{d}\}}},\frac{(s,v,o)\in q_\sigma}{q_{\sigma\cup\{v\to\hookrightarrow_{r}\}}} \tag{1}$$

$$\frac{(\mathtt{s},\mathtt{p},\mathtt{o})\in q_\sigma,(\mathtt{s},\mathtt{p},\mathtt{o})\in O}{q_\sigma[(\mathtt{s},\mathtt{p},\mathtt{o})/-]} \tag{2}$$

$$\frac{(v_1,\hookleftarrow,v_2)\in q_\sigma,(\mathtt{p},\hookleftarrow,\mathtt{c})\in O}{q_\sigma[(v_1,\hookleftarrow,v_2)/(v_1,\prec_{sp},\mathtt{p}),(\mathtt{c},\prec_{sc},v_2)]} \tag{3}$$

$$\frac{(v_1,\hookleftarrow,v_2)\in q_\sigma,(\mathtt{p},\hookleftarrow,\mathtt{c})\in O}{q_{\sigma\cup\{v_1\to\mathtt{p}\}}} \tag{4}$$

$$\frac{(v_1,\hookleftarrow,v_2)\in q_\sigma,(\mathtt{p},\hookleftarrow,\mathtt{c})\in O}{q_{\sigma\cup\{v_2\to\mathtt{c}\}}} \tag{5}$$

$$\frac{(v,\hookleftarrow,\mathtt{c})\in q_\sigma,(\mathtt{p},\hookleftarrow,\mathtt{c})\in O}{q_{\sigma\cup\{v\to\mathtt{p}\}}} \tag{6}$$

$$\frac{(\mathtt{p},\hookleftarrow,v)\in q_\sigma,(\mathtt{p},\hookleftarrow,\mathtt{c})\in O}{q_{\sigma\cup\{v\to\mathtt{c}\}}} \tag{7}$$

$$\frac{(v,\hookleftarrow,\mathtt{c})\in q_\sigma,(\mathtt{p},\hookleftarrow,\mathtt{c})\in O}{q_\sigma[(v,\hookleftarrow,\mathtt{c})/(v,\prec_{sp},\mathtt{p})]} \tag{8}$$

$$\frac{(\mathtt{p},\hookleftarrow,v)\in q_\sigma,(\mathtt{p},\hookleftarrow,\mathtt{c})\in O}{q_\sigma[(\mathtt{p},\hookleftarrow,v)/(\mathtt{c},\prec_{sc},v)]} \tag{9}$$

$$\frac{(s,\hookleftarrow,\mathtt{c}_1)\in q_\sigma,(\mathtt{c},\prec_{sc},\mathtt{c}_1)\in O,\mathtt{c}\neq\mathtt{c}_1}{q_\sigma[(s,\hookleftarrow,\mathtt{c}_1)/(s,\hookleftarrow,\mathtt{c})]} \tag{10}$$

$$\frac{(\mathtt{p},\hookleftarrow,o)\in q_\sigma,(\mathtt{p},\prec_{sp},\mathtt{p}_1)\in O,\mathtt{p}\neq\mathtt{p}_1}{q_\sigma[(\mathtt{p},\hookleftarrow,o)/(\mathtt{p}_1,\hookleftarrow,o)]} \tag{11}$$

$$\frac{(v_1,\prec,v_2)\in q_\sigma,(\mathtt{c}_1,\prec,\mathtt{c}_2)\in O}{q_{\sigma\cup\{v_1\to\mathtt{c}_1\}}} \tag{12}$$

$$\frac{(v,\prec,\mathtt{c}_2)\in q_\sigma,(\mathtt{c}_1,\prec,\mathtt{c}_2)\in O}{q_{\sigma\cup\{v\to\mathtt{c}_1\}}} \tag{13}$$

$$\frac{(\mathtt{c}_1,\prec,v)\in q_\sigma,(\mathtt{c}_1,\prec,\mathtt{c}_2)\in O}{q_{\sigma\cup\{v\to\mathtt{c}_2\}}} \tag{14}$$

$$\frac{(\mathtt{c}_1,\prec,o)\in q_\sigma,(\mathtt{c}_1,\prec,\mathtt{c}_2)\in O,\mathtt{c}_1\neq\mathtt{c}_2}{q_\sigma[(\mathtt{c}_1,\prec,o)/(\mathtt{c}_2,\prec,o)]} \tag{15}$$

$$\frac{(s,\prec,\mathtt{c}_2)\in q_\sigma,(\mathtt{c}_1,\prec,\mathtt{c}_2)\in O,\mathtt{c}_1\neq\mathtt{c}_2}{q_\sigma[(s,\prec,\mathtt{c}_2)/(s,\prec,\mathtt{c}_1)]} \tag{16}$$

Fig. 1. Reformulation rules for a partially instantiated query q_σ w.r.t. an RDFS ontology O. For compactness, we factorize similar rules, using the symbol $\hookleftarrow$ to denote either $\hookleftarrow_d$ or $\hookrightarrow_r$, and $\prec$ to denote either $\prec_{sc}$ or $\prec_{sp}$.

However, such a rule is flawed: it would blindly consider all triples $(\mathsf{p}, \hookleftarrow, \mathsf{c})$ from O, which causes a combinatorial explosion. Instead, we propose Rules (10) and (11), which use p' and c' as guides to replace $(\mathsf{p}', \hookleftarrow, \mathsf{c}')$ by other domain/range triples based on the subproperty-chains from p' and the subclass-chains to c'.

Query triples with a subclass ($\prec_{sc}$) or subproperty ($\prec_{sp}$) property are processed by Rules (12)–(16). Rules (12), (13), (14) instantiate a variable using an ontology triple of the form $(\mathsf{c}_1, \prec, \mathsf{c}_2)$. In Rule (12), which considers a query triple with two variables and instantiates one of these variables, we arbitrarily chose to instantiate the first variable. The two last rules allow to go up or down in the class and property hierarchies.

4.3 Reformulation Algorithm Associated with $\mathcal{R}_c$

The reformulation algorithm itself, denoted by $\text{Reformulate}_\text{c}$, is presented in Algorithm 1. The set of queries to be explored (named *toExplore*) initially contains q. Exploring a query consists of generating all new queries that can be obtained from it by applying a reformulation rule (lines 7–9). Newly generated queries are put in the set named *produced*. The algorithm proceeds in a breadth-first manner, exploring at each step the queries that have been generated at the previous step. When no new query can be generated at a step, the algorithm stops, otherwise the next step will explore the newly generated queries (line 11). Note the use of a set named *explored*, which contains all explored queries; the purpose of this set is to avoid infinite generation of the same queries when the subclass or subproperty hierarchy contains cycles (other than loops), otherwise it is useless. Importantly, not all explored queries are returned in the resulting set, but only those that no longer contain RDFS triples (lines 5–6). Indeed, on the one hand RDFS triples that contain variables are instantiated by the rules in all possible ways using the ontology, and, on the other hand, instantiated triples that belong to the ontology are removed (by Rule (2)). Finally, note that a variable v in a triple of the form (s, v, o) is replaced by a built-in RDFS property in some queries (by Rule (1)) and left unchanged in others as it may also be later mapped to a user-defined property in the RDF graph G.

A simple analysis of the reformulation rule behavior shows that the worst-case time complexity of algorithm $\text{Reformulate}_\text{c}$ is polynomial in the size of O and simply exponential in the size of q. More precisely:

Proposition 1. *The algorithm* $\text{Reformulate}_\text{c}$ *runs in time* $\mathcal{O}(|Val(O)|^{6|q|})$, *where* $|q|$ *is the number of triples in the body of* q.

The correctness of the algorithm relies on the following lemma, which characterizes the saturated graph $G^{\mathcal{R}_c}$ from the triples of G. We call $\prec_{sc}$-chain (resp. $\prec_{sp}$-chain) from s to o a possibly empty sequence of triples $(s_i, \prec_{sc}, o_i)$ (resp. $(s_i, \prec_{sp}, o_i)$) with $1 \leq i \leq n$, such that $s_1 = s$, $o_n = o$ and, for $i > 1$, $s_i = o_{i-1}$.

Algorithm 1. Reformulate$_\mathrm{c}$

```
Input  : BGPQ $q$ and ontology $O$
Output: the reformulation of $q$ with the rules from Fig. 1
1  $result \leftarrow \emptyset$; $toExplore \leftarrow \{q\}$; $explored \leftarrow \emptyset$
2  while $toExplore \neq \emptyset$ do
3      $produced \leftarrow \emptyset$
4      for each $q_\sigma \in toExplore$ do
5          if $q_\sigma$ does not contain any RDFS triple then
6              $result \leftarrow result \cup \{q_\sigma\}$
7          for each RDFS triple $t$ in $q_\sigma$ do
8              for each $q'_\sigma$ obtained by applying a reformulation rule to $t$ do
9                  $produced \leftarrow produced \cup \{q'_\sigma\}$
10         $explored \leftarrow explored \cup \{q_\sigma\}$
11     $toExplore \leftarrow produced \setminus explored$
12 return $result$
```

Since we do not enforce the reflexivity of the subclass relation, a triple $(\mathsf{c}, \prec_{sc}, \mathsf{c})$ belongs to $G^{\mathcal{R}}$ if and only if there is a non-empty $\prec_{sc}$-chain from c to c (which includes the case $(\mathsf{c}, \prec_{sc}, \mathsf{c}) \in G$). The same holds for the subproperty relation.

Lemma 1. *Let G be an RDF graph. It holds that:*

- $(\mathsf{c}, \prec_{sc}, \mathsf{c}') \in G^{\mathcal{R}_c}$ *iff G contains a non-empty $\prec_{sc}$-chain from* c *to* c'*;*
- $(\mathsf{p}, \prec_{sp}, \mathsf{p}') \in G^{\mathcal{R}_c}$ *iff G contains a non-empty empty $\prec_{sp}$-chain from* p *to* p'*;*
- $(\mathsf{p}', \hookleftarrow_d, \mathsf{c}') \in G^{\mathcal{R}_c}$ *iff G contains a triple $(\mathsf{p}, \hookleftarrow_d, \mathsf{c})$, a (possibly empty) $\prec_{sp}$-chain from p' to p and a (possibly empty) $\prec_{sc}$-chain from c to c'. The case for $(\mathsf{p}', \hookrightarrow_r, \mathsf{c}') \in G^{\mathcal{R}_c}$ is similar (replace $\hookleftarrow_d$ by $\hookrightarrow_r$ in the statement above).*

Below, we assume *without loss of generality* that the input query does not contain blank nodes; if needed, these have been equivalently replaced by variables. Therefore, all blank nodes that occur in the output reformulation have been introduced by the reformulation rules, and specifically refer to unknown classes and properties they identify within the ontology at hand. This justifies the subsequent use of non-standard query evaluation and answering in the next theorems.

Theorem 1. *Let G be an RDF graph with ontology O and q be a BGP query without blank nodes. Let $\mathcal{Q}_c$ be the output of* Reformulate$_\mathrm{c}(q, O)$*. Then:*

$$q(G, \mathcal{R}_c) = \widehat{q(G, \mathcal{R}_c)} = \widehat{\mathcal{Q}_c(G)} \tag{18}$$

$$q(G, \mathcal{R}_c \cup \mathcal{R}_a) = \widehat{q(G, \mathcal{R}_c \cup \mathcal{R}_a)} = \widehat{\mathcal{Q}_c(G, \mathcal{R}_c \cup \mathcal{R}_a)} \tag{19}$$

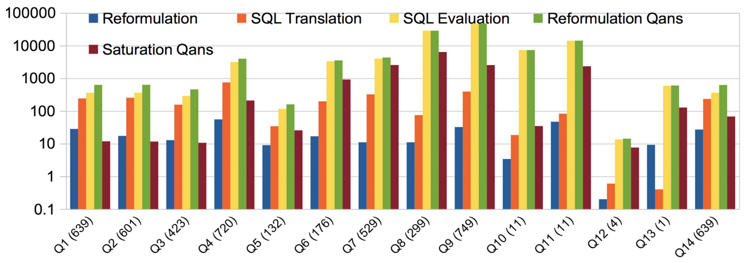

Fig. 2. Query answering times through reformulation and saturation.

Example 5. Consider the BGPQ asking for *how someone is related to some particular kind of company*: $q(x, y) \leftarrow (x, y, z), (z, \tau, t), (y, \prec_{sp}, \text{:worksFor}), (t, \prec_{sc},$:Comp). Its answer set on G_{ex} w.r.t. $\mathcal{R}$, which can be easily checked using $(G_{\text{ex}})^{\mathcal{R}}$ provided in Sect. 2, is: $q(G_{\text{ex}}, \mathcal{R}) = \{\langle \text{:p1}, \text{:ceoOf}\rangle\}$.

The output of $\text{Reformulate}_c(q, \text{RDFS}(G_{\text{ex}}))$ is:

$$\begin{aligned}\mathcal{Q}_c = & \{q'(x, \text{:ceoOf}) \leftarrow (x, \text{:ceoOf}, z), (z, \tau, \_\text{:}b_C),\\ & q''(x, \text{:hiredBy}) \leftarrow (x, \text{:hiredBy}, z), (z, \tau, \_\text{:}b_C)\}\end{aligned}$$

where q' and q'' are obtained by binding, using Rule (13), y to either :ceoOf or :hiredBy, and t to $\_\text{:}b_C$. Further, these bindings have also produced the fully instantiated RDFS constraints $(\text{:ceoOf}, \prec_{sp}, \text{:worksFor})$ and $(\_\text{:}b_C, \prec_{sc}, \text{:Comp})$ in q', as well as $(\text{:hiredBy}, \prec_{sp}, \text{:worksFor})$ and $(\_\text{:}b_C, \prec_{sc}, \text{:Comp})$ in q'', which have then been eliminated by Rule (2).

The *non-standard* answering of $\mathcal{Q}_c$ on G_{ex} w.r.t. $\mathcal{R}$, i.e., $\overbrace{q'(G_{\text{ex}}, \mathcal{R})} \cup \overbrace{q''(G_{\text{ex}}, \mathcal{R})}$ provides the correct answer set $\{\langle \text{:p}_1, \text{:ceoOf}\rangle\}$, whose only tuple results from q'. Using *standard* answering, the incorrect answer $\langle \text{:p}_2, \text{:hiredBy}\rangle$ would have also been obtained from q'', since under this semantics q'' asks for *who is hired by an organization of some type* (this is the case of :p_2 who is hired by a public administration) and not *who is hired by an organization of the particular unknown type of company designated by* $\_\text{:}b_C$ *in* G_{ex}.

We now rely on the query reformulation algorithm, from [13], say Reformulate_a, which takes as input a partially instantiated BGPQ q *without RDFS triples*, and a graph G and, using a set of reformulation rules associated with $\mathcal{R}_a$, outputs a reformulation $\mathcal{Q}_a$ such that: $q(G, \mathcal{R}_c \cup \mathcal{R}_a) = q(G, \mathcal{R}_a) = \overbrace{\mathcal{Q}_a(G)}$. The adaptation of Reformulate_a to an input UBGPQ instead of a BGPQ is straightforward. Furthermore, we notice that the algorithm would consider potential blank nodes in the input query as if they were IRIs. Hence, denoting by $\mathcal{Q}_{c,a}$ the output of $\text{Reformulate}_a(\mathcal{Q}_c, O)$, we obtain:

$$\overbrace{\mathcal{Q}_c(G, \mathcal{R}_c \cup \mathcal{R}_a)} = \overbrace{\mathcal{Q}_c(G, \mathcal{R}_a)} = \overbrace{\mathcal{Q}_{c,a}(G)} \qquad (20)$$

Putting together (20) and statement (19) in Theorem 1, we can prove the correctness of the global reformulation algorithm:

Theorem 2. *Let G be an RDF graph and q be a BGPQ without blank nodes. Let $\mathcal{Q}_{c,a}$ be the reformulation of q by the 2-step algorithm described in Sect. 4.1. Then: $q(G, \mathcal{R}_c \cup \mathcal{R}_a) = \overbrace{\mathcal{Q}_{c,a}(G)}$ holds.*

5 Experimental Evaluation

We have implemented our reformulation algorithm on top of OntoSQL (https://ontosql.inria.fr), a Java platform providing efficient RDF storage, saturation, and query evaluation on top of an RDBMS [10,13]; we used Postgres v9.6. To save space, OntoSQL encodes IRIs and literals into integers, and a dictionary table which allows going from one to the other. It stores all resources of a certain type in a one-attribute table, all subject, object pairs for each data property in a table, and all schema triples in another table; the tables are indexed. Our server has a 2,7 GHz Intel Core i7 and 160 GB of RAM; it runs CentOs Linux 7.5.

We generated LUBM$^\exists$ data graphs [16] of 10M triples and restricted the ontology to RDFS, leading to 175 triples (123 $\prec_{sc}$, 5 $\prec_{sp}$, 25 $\hookleftarrow_d$ and 22 $\hookrightarrow_r$). We devised 14 queries having from 3 to 7 triples; one has no result, while the others have a few dozen to three hundred thousand results. Each has 1 or 2 triples which match the ontology (and must be evaluated on it for correctness), including (but not limited to) the generic triple (x, y, z), which appears 7 times overall in our workload. Some of our queries are not handled through reformulation by AllegroGraph and Stardog, nor by Virtuoso (recall Sect. 3).

Figure 2 shows for each query: the size of the UBGPQ reformulation (in parenthesis after the query name on the x axis), i.e., the number of BGPQs it contains; the reformulation time (with both $\mathcal{R}_c$ and $\mathcal{R}_a$); the time to translate the reformulation into SQL; the time to evaluate this SQL query; the total query answering time through reformulation, and (for comparison) through saturation. Note the logarithmic y axis. *Details of our experiments are available online* (see Footnote 1). The reformulation time is very short (0.2 ms to 55 ms). Unsurprisingly, the time to convert the reformulation into SQL is closely correlated with the reformulation size. The overhead of our approach is quite negligible, given that the answering time through reformulation is close to the SQL evaluation time.

As expected, saturation-based query answering is faster; however, saturating this graph took more than 1289 s, while the slowest query (Q9) took 46 s. As in [13], we compute for each query Q a *threshold* n_Q which is the smallest number of times we need to run Q, so that saturating G and running Q n_Q times on $G^{\mathcal{R}}$ is faster than n_Q runs of Q through reformulation; intuitively, *after n_Q runs of Q, the saturation cost amortizes.* For our queries, n_Q ranged from 29 (Q9) to 9648 (Q5), which shows that saturation costs take a while to amortize. If the graph or the ontology change, requiring maintenance of the saturated graph, reformulation may be even more competitive.

6 Conclusion

We have presented a novel reformulation-based query answering technique for RDF graphs with RDFS ontologies. Its novelty lies in its capacity to handle query triples over both the assertions and the ontology; such queries are not always handled correctly by existing RDF engines. In the future, we plan to integrate our reformulation technique in the cost-based optimized reformulation framework we introduced in [10] to improve its performance, and to an OBDA setting along the lines of [15].

Acknowledgements. This work is supported by the Inria Project Lab grant iCoda, a collaborative project between Inria and several major French media.

References

1. RDF 1.1 Concepts and Abstract Syntax. https://www.w3.org/TR/rdf11-concepts
2. RDF 1.1 Semantics. https://www.w3.org/TR/rdf11-mt/#rdfs-entailment
3. SPARQL 1.1 Query Language. https://www.w3.org/TR/sparql11-query/
4. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
5. Adjiman, P., Goasdoué, F., Rousset, M.C.: SomeRDFS in the semantic web. JODS **8**, 158–181 (2007)
6. Arenas, M., Gutierrez, C., Pérez, J.: Foundations of RDF databases. In: Tessaris, S., Franconi, E., Eiter, T., Gutierrez, C., Handschuh, S., Rousset, M.-C., Schmidt, R.A. (eds.) Reasoning Web 2009. LNCS, vol. 5689, pp. 158–204. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03754-2_4
7. Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z., Velkov, R.: OWLIM: a family of scalable semantic repositories. Semant. Web **2**(1), 33–42 (2011)
8. Broekstra, J., Kampman, A.: Inferencing and truth maintenance in RDF schema. In: PSSS1 Workshop (2003). http://ceur-ws.org/Vol-89/broekstra-et-al.pdf
9. Buron, M., Goasdoué, F., Manolescu, I., Mugnier, M.L.: Reformulation-based query answering for RDF graphs with RDFS ontologies. Research report, Inria, March 2019. https://hal.archives-ouvertes.fr/hal-02051413
10. Bursztyn, D., Goasdoué, F., Manolescu, I.: Optimizing reformulation-based query answering in RDF. In: EDBT (2015)
11. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: the DL-Lite family. J. Autom. Reasoning (JAR) **39**(3), 385–429 (2007)
12. Goasdoué, F., Karanasos, K., Leblay, J., Manolescu, I.: View selection in semantic web databases. PVLDB **5**(2) (2011). https://hal.inria.fr/inria-00625090v1
13. Goasdoué, F., Manolescu, I., Roatis, A.: Efficient query answering against dynamic RDF databases. In: EDBT (2013). https://hal.inria.fr/hal-00804503v2
14. Kaoudi, Z., Miliaraki, I., Koubarakis, M.: RDFS reasoning and query answering on top of DHTs. In: Sheth, A., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 499–516. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88564-1_32

15. Lanti, D., Xiao, G., Calvanese, D.: Cost-driven ontology-based data access. In: d'Amato, C., et al. (eds.) ISWC 2017. LNCS, vol. 10587, pp. 452–470. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68288-4_27
16. Lutz, C., Seylan, İ., Toman, D., Wolter, F.: The combined approach to OBDA: taming role hierarchies using filters. In: Alani, H., et al. (eds.) ISWC 2013. LNCS, vol. 8218, pp. 314–330. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41335-3_20
17. Neumann, T., Weikum, G.: The RDF-3X engine for scalable management of RDF data. VLDB J. **19**, 91–113 (2010)
18. Urbani, J., van Harmelen, F., Schlobach, S., Bal, H.: QueryPIE: backward reasoning for OWL Horst over very large knowledge bases. In: Aroyo, L., et al. (eds.) ISWC 2011. LNCS, vol. 7031, pp. 730–745. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25073-6_46
19. Urbani, J., Piro, R., van Harmelen, F., Bal, H.E.: Hybrid reasoning on OWL RL. Semant. Web **5**(6), 423–447 (2014)