BGP Query Answering against Dynamic RDF Databases

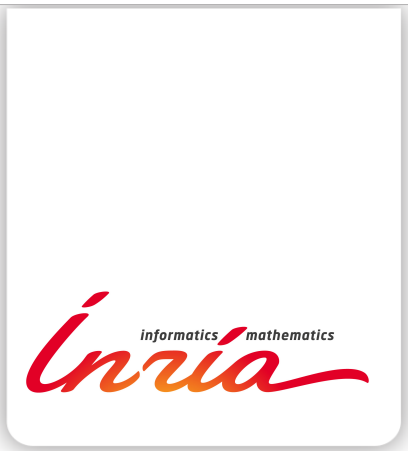
François Goasdoué, Ioana Manolescu, Alexandra Roatis

BGP Query Answering against Dynamic RDF Databases

François Goasdoué, Ioana Manolescu, Alexandra Roatiş

RESEARCH REPORT
N° 8018
July 2012
Project-Team OAK

ISSN 0249-6399 ISRN INRIA/RR--8018--FR+ENG



BGP Query Answering against Dynamic RDF Databases

François Goasdoué*, Ioana Manolescu*, Alexandra Roatiş*

Project-Team OAK

Research Report n° 8018 — July 2012 — 42 pages

Abstract:
A promising method for efficiently querying RDF data consists of translating SPARQL queries into efficient RDBMS-style operations. However, answering SPARQL queries requires handling *RDF reasoning*, which must be implemented outside the relational engines that do not support it.
We introduce the expressive *database (DB) fragment of RDF* for which we devise novel *sound and complete techniques* for answering *Basic Graph Pattern (BGP)* queries. Our techniques explore the two established approaches for handling RDF semantics, namely reformulation and saturation; we show how they cope with *updates*, a complex problem due to the rich RDF semantics. Our algorithms can be deployed on top of any RDBMS(-style) engine, and we experimentally study their performance trade-offs.

Key-words: RDF fragments, query answering, reasoning

* Université Paris-Sud and INRIA Saclay

RESEARCH CENTRE
SACLAY – ÎLE-DE-FRANCE

Parc Orsay Université
4 rue Jacques Monod
91893 Orsay Cedex

Répondre aux requêtes BGP sur des bases de données RDF dynamiques

Résumé : Une méthode prometteuse pour l'interrogation efficace de données RDF consiste à traduire les requêtes SPARQL en opérations typiques des SGBDR. Toutefois, répondre aux requêtes SPARQL nécessite de tenir compte de la *sémantique de RDF*, ce qui implique d'utilisation de mécanismes de raisonnement en dehors des moteurs relationnels.

Nous introduisons le fragment *base de données (DB)* de RDF, pour lequel nous étudions de nouvelles techniques *exactes* pour répondre aux requêtes de type Basic Graph Pattern (BGP). Nos techniques couvrent les deux approches de raisonnement bien établies pour la prise en compte de la sémantique de RDF, c'est-à-dire la reformulation de requêtes et la saturation des données ; nous montrons comment elles s'adaptent aux mises à jour, un problème complexe du fait de la sémantique de RDF. Nos algorithmes peuvent être mis en œuvre au-dessus de tout moteur de type SGBDR, et nous comparons expérimentalement leurs performances.

Mots-clés : fragments RDF, réponses aux requêtes, raisonnement

Contents

1 Introduction

The Resource Description Framework (RDF) [21] is a graph-based data model accepted as the W3C standard for Semantic Web applications. As such, it comes with an ontology language, *RDF Schema* (RDFS), that can be used to enhance the description of RDF *graphs*, i.e., RDF datasets. The W3C standard for querying RDF is SPARQL Protocol and RDF Query Language (SPARQL) [20].

The literature provides several scalable solutions for querying RDF graphs using relational data management systems (RDBMSs, in short) or RDBMS-style specialized engines [1, 17, 22]. These works, however, ignore the essential RDF feature called *entailment*, which allows modeling *implicit information* within RDF. Taking entailment into account is crucial for answering SPARQL queries, as ignoring implicit information leads to incomplete answers [20]. Thus, to capitalize on (and benefit from) scalable RDBMS performance, SPARQL query

answering can be split into a *reasoning* step which handles entailment outside the RDBMSs, and a *query evaluation* step delegated to RDBMSs.

A popular reasoning step is *graph saturation* (a.k.a. *closure*). This consists of pre-computing (making explicit) and adding to the RDF graph all implicit information. Answering queries using saturation amounts to evaluating the queries against the saturated graph. While saturation leads to efficient query processing, it requires time to be computed, space to be stored, and must be re-computed upon updates. The alternative reasoning step is *query reformulation*. This consists in turning the query into a *reformulated query*, which, evaluated against the original graph, yields the correct answers to the original query. Since reformulation is made at query run-time, it is intrinsically robust to updates; reformulation is also typically very fast. However, reformulated queries are often syntactically more complex than the original ones, thus their evaluation may be costly.

RDF on one hand, and SPARQL on the other hand, are complex languages with many features. For instance, the RDF specification supports a form of *incomplete information* through *blank nodes* and it provides a large set of *entailment rules* for deriving implicit information. The forthcoming SPARQL 1.1 supports aggregates, negation etc.

If saturation is used, one first choses an RDF fragment and saturates the RDF graph accordingly. Then in a fully orthogonal way, one choses the SPARQL dialect to evaluate on the graph thus saturated. In contrast, reformulation leads to a subtle interplay between the RDF and SPARQL dialects, since the query must be reformulated within the latter, so as to compute correct query answers with respect to the former. Reformulation-based query answering has been studied for the Description Logics (DL) [5] fragment of RDF and the relational conjunctive SPARQL subset [3, 8, 11], and extensions thereof [4, 10, 15, 19].

In this paper, we devise and compare *query answering techniques* for the *database (DB) fragment of RDF* and the *Basic Graph Pattern* (BGP) queries of SPARQL. This DB fragment, which we introduce, extends the fragments mentioned above, notably with the support of RDF *blank nodes*, encoding incomplete information. The *Basic Graph Pattern* (BGP) queries, identified in the SPARQL recommendation, are more expressive than relational conjunctive queries, since BGPs also allow *querying (and joining on) RDF relation names (called classes and properties)*, which conjunctive queries do not allow.

Our contributions within this DB fragment are:

1. a saturation-based query answering technique robust to updates, relying on saturation maintenance;
2. a reformulation-based query answering technique;
3. a thorough experimental study of the performance trade-offs between the two techniques.

Importantly, our techniques can work on top of *any standard RDBMS*, to which we delegate RDF graphs storage and BGP query evaluation (through SQL), while implementing out of the core all the necessary steps in order to compute correct answers. This allows extending the benefits of RDBMS scalability and reliability to a larger RDF fragment (the DB fragment) than in previous works. The positioning of our work with respect to the most prominent previous ones is sketched in Figure 1 (more details are provided in Section 9).

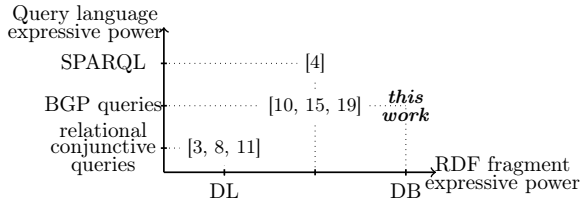


Figure 1: Outline of the positioning of our work.

The paper is organized as follows. Section 2 introduces RDF graphs and BGP queries. Section 3 relates RDF to relational databases. Section 4 defines our DB fragment of RDF, for which Section 5 presents saturation-based and reformulation-based query answering techniques. These techniques are then studied in-depth in Sections 6 and 7, and experimentally compared in Section 8. We discuss related work in Section 9, then we conclude.

2 RDF graphs and queries

We introduce *RDF graphs*, modeling RDF datasets, in Section 2.1, and *Basic Graph Pattern queries* in Section 2.2.

2.1 RDF Graphs

An *RDF graph* (or *graph*, in short) is a set of *triples* of the form `s p o .` (the final dot preceded by a white space belongs to the normative triple syntax). A triple states that its *subject* `s` has the *property* `p`, and the value of that property is the *object* `o`. Given a set U of URIs, a set L of literals (constants), and a set B of blank nodes (unknown URIs or literals), such that U, B and L are pairwise disjoint, a triple is *well-formed* whenever its subject belongs to $U \cup B$, its property belongs to U, and its object belongs to $U \cup B \cup L$. In what follows, we only consider well-formed triples.

Blank nodes are essential features of RDF allowing to support *unknown URI/literal tokens*. For instance, one can use a blank node _:b_1 to state that the country of _:b_1 is *United States* while the city of the same _:b_1 is *Washington*. Many such blank nodes can co-exist within a graph, e.g., one may also state that the country of _:b_2 is *Sweden* while the city of _:b_2 is *Stockholm*; at the same time, the population of *Stockholm* can be said to be an unspecified value _:b_3.

Notations. We use `s`, `p`, `o` and _:b in triples (possibly with subscripts) as placeholders. That is, `s` stands for values in $U \cup B$, `p` stands for values in U, `o` represents values from $U \cup B \cup L$, and _:b denotes values in B. Finally, we use strings between quotes as in "*string*" to denote literals.

Figure 2 shows how to use triples to describe resources; we use the usual namespaces rdf: and rdfs: for classes and properties comprised in the RDF standard [21], and which can be seen as an RDF meta-model. rdf:type is used to specify to which *classes* a resource belongs. This can be seen as a form of resource typing.

Constructor	Triple	Relational notation
Class assertion	s rdf:type o .	o(s)
Property assertion	s p o .	p(s, o)

Figure 2: RDF statements.

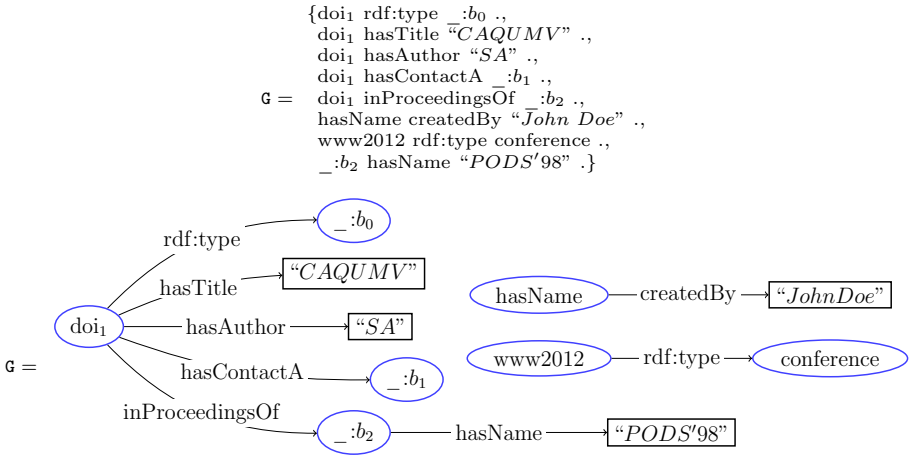


Figure 3: Alternative RDF graph representations.

A more intuitive representation of a *graph* can be drawn from its triples: every distinct subject or object value corresponds to a node labeled with this value; for each triple, there is a directed edge from the subject node to the object one, labeled with the property value.

Example 1 (Running example)**.** *The two representations in Figure 3 are equivalent as they model the same graph* G*. The namespaces for user-defined classes and properties were omitted for the sake of readability. This graph describes the resource* doi_1 *that belongs to an unknown class, whose title (*hasTitle*) is "Complexity of Answering Queries Using Materialized Views", whose author (*hasAuthor*) is "Serge Abiteboul" and which has an unknown contact author (*hasContactA*). This paper is in the proceedings of (*inProceedingsOf*) an unknown resource whose name (*hasName*) is "PODS'98". Lastly, the URI* www2012 *is a conference and the property that associates names (*hasName*) has been created by "John Doe".*

A valuable feature of RDF is RDF Schema (RDFS) that allows enhancing the descriptions in graphs. An RDF Schema declares *semantic constraints* between the classes and the properties used in these graphs. Figure 4 shows the allowed constraints and how to express them; in this figure, $\mathtt{s}, \mathtt{o} \in U \cup B$, while *domain* and *range* denote respectively the first and second attribute of every property. The figure also relates these constraints to relational inclusion constraints under the open-world assumption.

Traditionally, constraints can be interpreted in two ways [2]: under the closed-world assumption (CWA) or under the open-world assumption (OWA). Under CWA, any fact not present in the database is assumed not to hold. Under this assumption, if the set of database facts does not respect a constraint,

Constructor	Triple	Relational notation
Subclass constraint	s rdfs:subClassOf o .	$s \subseteq o$
Subproperty constraint	s rdfs:subPropertyOf o .	$s \subseteq o$
Domain typing constraint	s rdfs:domain o .	$\Pi_{\text{domain}}(s) \subseteq o$
Range typing constraint	s rdfs:range o .	$\Pi_{\text{range}}(s) \subseteq o$

Figure 4: RDFS statements.

$G' = G \cup$ {posterCP rdfs:subClassOf confP .,
_:b_0 rdfs:subClassOf confP .,
confP rdfs:subClassOf paper .,
hasTitle rdfs:domain paper .,
hasTitle rdfs:range rdfs:Literal .,
hasAuthor rdfs:domain paper .,
hasAuthor rdfs:range rdfs:Literal .,
hasContactA rdfs:subPropertyOf hasAuthor .,
inProceedingsOf rdfs:domain confP .,
inProceedingsOf rdfs:range conference .,
hasName rdfs:domain conference .,
hasName rdfs:range rdfs:Literal .,
createdBy rdfs:range rdfs:Literal .}

Figure 5: Extended RDF graph for Example 2.

then the database is *inconsistent*. For instance, the CWA interpretation of a constraint of the form $R_1 \subseteq R_2$ is: any tuple in the relation R_1 *must* also be in the relation R_2 *in the database*, otherwise the database is inconsistent. On the contrary, under OWA, some facts may hold even though they are *not in the database*. For instance, the OWA interpretation of the same example is: any tuple t in the relation R_1 *is considered as being also in the relation* R_2 (the inclusion constraint *propagates* t *to* R_2).

The RDF data model – and accordingly, the present work – is based on OWA, and this is how we interpret all the constraints in Figure 4. To give another example in RDF, if the triples hasFriend rdfs:domain Person . and Anne hasFriend Marie . hold in the graph, then so does the triple Anne rdf:type Person . The latter is due to the rdfs:domain constraint in Figure 4.

Example 2 (Continued)**.** *Consider next to the above graph* G*, a schema stating that poster papers (*posterCP*) together with the unknown class* _:b_0 *of which* doi_1 *is an instance, are subclasses of conference papers (*confP*), which are scientific papers (*paper*). Moreover, titles (*hasTitle*), authors (*hasAuthor*), contact authors (*hasContactA*) (who are authors) are used to describe papers, which are connected to the conferences (*conference*) in whose proceedings (*inProceedingsOf*) they appear. Finally, names (*hasName*) describe conferences, and creators (*createdBy*) describe resources. The extended graph* G′ *of* G *corresponding to this schema is depicted in Figure 5.*

Entailment. Our discussion about OWA above illustrated an important RDF feature: *implicit triples*, considered to be part of the graph even though they are not explicitly present in it. The W3C names *entailment* the mechanism through which, based on the set of explicit triples and some *entailment rules* (to be described shortly), implicit RDF triples are derived. We denote by $\vdash^{i}_{\text{RDF}}$ *immediate entailment*, i.e., the process of deriving new triples through a single application of an entailment rule. Then, (full) *RDF entailment* can be defined

as follows: a triple `s p o .` is entailed by a graph `G`, denoted $\texttt{G} \vdash_{\text{RDF}} \texttt{s p o .}$ if and only if there is a sequence of applications of immediate entailment rules that leads from `G` to `s p o .` (where at each step of the entailment sequence, the triples previously entailed are also taken into account).

Graph saturation. The immediate entailment rules allow defining the finite *saturation* (a.k.a. *closure*) of a graph `G`, which is the graph, denoted $\texttt{G}^\infty$, defined as the fixpoint of:

- $\texttt{G}^0 = \texttt{G}$
- $\texttt{G}^\alpha = \texttt{G}^{\alpha-1} \cup \{\texttt{s p o .} \mid \texttt{G}^{\alpha-1} \vdash^i_{\text{RDF}} \texttt{s p o .}\}$

The saturation of a graph is unique (up to blank node renaming), and does not contain any implicit triples (they have all been made explicit by saturation). An obvious connection holds between the triples entailed by a graph `G` and its saturation: $\texttt{G} \vdash_{\text{RDF}} \texttt{s p o .}$ if and only if $\texttt{s p o .} \in \texttt{G}^\infty$. It is worth noticing that entailment is part of the RDF specification itself, and therefore the semantics of a graph is its saturation. From the RDF standard perspective, any graph `G` is equivalent to, and models, its saturation $\texttt{G}^\infty$.

Immediate entailment rules. We give here an overview of the different kinds of immediate entailment rules upon which RDF entailment relies.

A first kind of rule generalizes triples using blank nodes. For instance, in the running example, doi$_1$ rdf:type _:b_0 . entails _:b_3 rdf:type _:b_0 . Indeed, if doi$_1$ is an instance of _:b_0, then there exists an instance of _:b_0.

A second kind of rule derives entailed triples from the semantics of built-in classes and properties. E.g., RDF provides rdfs:Class whose semantics is the set of all built-in *and* user-defined classes, with the striking effect that rdfs:Class is an instance of itself. As a result, in the running example, doi$_1$ rdf:type _:b_0 . entails that _:b_0 is a class, i.e., _:b_0 rdf:type rdfs:Class .

Finally, the third kind of rule derives entailed triples from the constraints modeled in an RDF Schema. Some rules derive entailed RDFS statements, through the transitivity of class and property inclusions, and from inheritance of domain and range typing. Using our running example, _:b_0 rdfs:subClassOf confP . *and* confP rdfs:subClassOf paper . entail _:b_0 rdfs:subClassOf paper .; and hasContactA rdfs:subPropertyOf hasAuthor . *and* hasAuthor rdfs:domain paper . entail hasContactA rdfs:domain paper . Some other rules derive entailed RDF statements, through the propagation values (URIs, blank nodes, and literals) from sub-classes and sub-properties to their super-classes and super-properties, and from properties to classes typing their domains and ranges. Using our running example, hasContactA rdfs:subPropertyOf hasAuthor . *and* doi$_1$ hasContactA _:b_1 . entail doi$_1$ hasAuthor _:b_1 .; and doi$_1$ hasAuthor _:b_1 . *and* hasAuthor rdfs:domain paper . entail doi$_1$ rdf:type paper .

2.2 BGP queries

We consider the well-known subset of SPARQL consisting of (unions of) basic graph pattern (BGP) queries.

A BGP is a *set of triple patterns*, or triples in short. Each triple has a subject, property and object. Subjects and properties can be URIs, blank nodes or variables; objects can also be literals.

We focus on the boolean BGP queries of the form `ASK WHERE` $\{t_1, \ldots, t_\alpha\}$, and on the non-boolean BGP queries of the form `SELECT` $\bar{x}$ `WHERE` $\{t_1, \ldots, t_\alpha\}$, where $\{t_1, \ldots, t_\alpha\}$ is a BGP, i.e., a set of triples modeling their conjunction; the variables $\bar{x}$ in the head of the query are called *distinguished variables*, and are a subset of the variables occurring in $t_1, \ldots, t_\alpha$.

Notations. Without loss of generality, in the following we will use the conjunctive query notation $q(\bar{x})$:- $t_1, \ldots, t_\alpha$ for both `ASK` and `SELECT` queries (for boolean queries, $\bar{x}$ is empty). We use x, y, and z (possibly with subscripts) to denote variables in queries. We denote by $\texttt{VarBl}(q)$ the set of variables *and* blank nodes occurring in the query q. The set of values (URIs, blank nodes, literals) of a graph `G` is denoted $\texttt{Val}(\texttt{G})$.

Query evaluation. Given a query q and a graph `G`, the *evaluation of* q *against* `G` is: $q(\texttt{G}) = \{\bar{x}_\mu \mid \mu : \texttt{VarBl}(q) \to \texttt{Val}(\texttt{G})$ is a total assignment s.t. $(t_1, \ldots, t_\alpha)_\mu \subseteq \texttt{G}\}$.

In the above, for any triple or set of triples O, we denote by O_μ the result of replacing every occurrence of a variable or blank node $e \in \texttt{VarBl}(q)$ in O by the value $\mu(e) \in \texttt{Val}(\texttt{G})$. If q is boolean, the empty answer set encodes false, while the non-empty answer set made of the empty tuple $\emptyset_\mu = \langle\rangle$ encodes true.

Notice that (normative) query evaluation *treats the blank nodes in a query as non-distinguished variables.* That is, one could consider equivalently queries without blank nodes or queries without non-distinguished variables.

Query answering. The evaluation of q against `G` only uses `G`'s explicit triples, thus may lead to an incomplete answer set. The (complete) *answer set* of q against `G` is obtained by the evaluation of q against $\texttt{G}^\infty$, denoted by $q(\texttt{G}^\infty)$.

Example 3 (Continued)**.** *The following query asks for the authors of papers published in the proceedings of a conference somehow related to PODS'98:*

$$\begin{aligned} q(x) \text{:-}\ & y_1 \text{ hasAuthor } x \text{ .}, \\ & y_1 \text{ inProceedingsOf } y_2 \text{ .}, \\ & y_2\ y_3 \text{ “}PODS'98\text{”} \text{ .} \end{aligned}$$

That query could be equivalently rewritten into:

$$\begin{aligned} q(x) \text{:-}\ & \_{:}b_0 \text{ hasAuthor } x \text{ .}, \\ & \_{:}b_0 \text{ inProceedingsOf } \_{:}b_1 \text{ .}, \\ & \_{:}b_1\ \_{:}b_2 \text{ “}PODS'98\text{”} \text{ .} \end{aligned}$$

The answer set of q against $\texttt{G}'$ is: $q(\texttt{G}'^\infty) = \{\langle$“$SA$”$\rangle, \langle \_{:}b_1 \rangle\}$. The answer “$SA$” *results from the assignment $\mu = \{y_1 \to \text{doi}_1, x \to$ “SA”$, y_2 \to \_{:}b_2, y_3 \to$ hasName$\}$, while the answer $\_{:}b_1$ results from $\texttt{G}' \vdash_{\text{RDF}} \text{doi}_1$ hasAuthor $\_{:}b_1$. and the assignment $\mu = \{y_1 \to \text{doi}_1, x \to \_{:}b_1, y_2 \to \_{:}b_2, y_3 \to$ hasName$\}$. Note that evaluating q against $\texttt{G}'$ leads to the incomplete answer set $q(\texttt{G}') = \{\langle$“$SA$”$\rangle\} \subset q(\texttt{G}'^\infty)$.*

3 RDF meets RDBMS

Graphs turn out to be a special case of *incomplete* relational databases based on *V-tables* [2, 14]. Such tables allow using variables in their tuples. Observe that repeating a variable within a V-table allows expressing joins on unknown values. An important result on V-table querying is that the standard relational

evaluation (which sees variables in V-tables as constants) computes the exact answer set of any conjunctive query [2, 14]. From a practical viewpoint, this provides a possible way of answering conjunctive queries against V-tables using standard relational database management systems (RDBMSs, in short).

From the above observations, a graph $\mathtt{G}$ can be encoded into a (single) V-table $\mathtt{Triple}(\mathtt{s},\mathtt{p},\mathtt{o})$ storing the triples of $\mathtt{G}$ as tuples, in which blank nodes become variables. Then, given a BGP query $q(\bar{x})$:- $\mathtt{s}_1\ \mathtt{p}_1\ \mathtt{o}_1\ .,\ldots,\mathtt{s}_n\ \mathtt{p}_n\ \mathtt{o}_n\ .$, in which blank nodes have been equivalently replaced by fresh non-distinguished variables, the SPARQL evaluation $q(\mathtt{G})$ of q against $\mathtt{G}$ is obtained by the relational evaluation of the conjunctive query $Q(\bar{x})$:- $\bigwedge_{i=1}^{n}\mathtt{Triple}(\mathtt{s}_i,\mathtt{p}_i,\mathtt{o}_i)$ against the aforementioned $\mathtt{Triple}$ table. Indeed, SPARQL and relational evaluations coincide with the above encoding, as relational evaluation amounts to finding all the total assignments from the variables of the query to the values (constants and variables) in the $\mathtt{Triple}$ table, so that the query becomes a subset of that $\mathtt{Triple}$ table.

It follows that evaluating $Q(\bar{x})$:- $\bigwedge_{i=1}^{n}\mathtt{Triple}(\mathtt{s}_i,\mathtt{p}_i,\mathtt{o}_i)$ against the $\mathtt{Triple}$ table containing the saturation of $\mathtt{G}$, instead of $\mathtt{G}$ itself, computes the answer set of q against $\mathtt{G}$. Hence, BGP queries against graphs can be evaluated by any RDBMS.

Example 4 (Continued)**.** *Provided that the saturation of the above graph* $\mathtt{G}'$ *is encoded into a V-table* $\mathtt{Triple}(\mathtt{s},\mathtt{p},\mathtt{o})$ *as described above, the answer set of the following BGP query:*

$$\begin{aligned} q(x)\text{:-}\ & y_1\ \text{hasAuthor}\ x\ .,\\ & y_1\ \text{inProceedingsOf}\ y_2\ .,\\ & y_2\ y_3\ \text{“}PODS'98\text{”}\ . \end{aligned}$$

against $\mathtt{G}'$ *(i.e.,* $q(\mathtt{G}'^{\infty})$*) is the same as the result of evaluating the following relational query against the* $\mathtt{Triple}$ *table:*

$$\begin{aligned} Q(x)\text{:-}\ & \mathtt{Triple}(y_1,\text{hasAuthor},x)\ \wedge\\ & \mathtt{Triple}(y_1,\text{inProceedingsOf},y_2)\ \wedge\\ & \mathtt{Triple}(y_2,y_3,\text{“}PODS'98\text{”}). \end{aligned}$$

4 The database fragment of RDF

We define a restriction of RDF that we call its *database (DB) fragment*. Our goal in identifying this fragment is to devise saturation- and reformulation-based query answering techniques that can be deployed on top of *any* off-the-shelf or RDF-tuned RDBMS. This DB fragment is obtained by:

- *restricting RDF entailment* [21] *to the rules dedicated to RDF Schema* only (a.k.a. RDFS entailment), shown in Figures 6–9;
- *not restricting graphs in any way*, in other words, any triple that the RDF specification allows is also allowed in the DB fragment.

We call a graph belonging to our DB fragment a *database*. A database $\mathtt{db}$ is a pair $\langle \mathtt{S},\mathtt{D}\rangle$, where $\mathtt{S}$ and $\mathtt{D}$ are two disjoint sets of triples. $\mathtt{S}$ triples can only be RDFS statements such as shown in Figure 4. We call these triples the

Triple	Entailed triple ($\vdash^i_{\text{RDF}}$)
$\mathtt{s}_1$ rdfs:subClassOf $\mathtt{s}_2$.	$\mathtt{s}_1$ rdfs:subClassOf $\mathtt{s}_1$.
$\mathtt{s}_1$ rdfs:subClassOf $\mathtt{s}_2$.	$\mathtt{s}_2$ rdfs:subClassOf $\mathtt{s}_2$.
$\mathtt{p}_1$ rdfs:subPropertyOf $\mathtt{p}_2$.	$\mathtt{p}_1$ rdfs:subPropertyOf $\mathtt{p}_1$.
$\mathtt{p}_1$ rdfs:subPropertyOf $\mathtt{p}_2$.	$\mathtt{p}_2$ rdfs:subPropertyOf $\mathtt{p}_2$.
p rdfs:domain s .	p rdfs:subPropertyOf p .
p rdfs:domain s .	s rdfs:subClassOf s .
p rdfs:domain rdfs:Literal .	p rdfs:subPropertyOf p .
p rdfs:range s .	p rdfs:subPropertyOf p .
p rdfs:range s .	s rdfs:subClassOf s .
p rdfs:range rdfs:Literal .	p rdfs:subPropertyOf p .

Figure 6: Schema-level entailment from a single schema-level triple.

Triple	Entailed triple ($\vdash^i_{\text{RDF}}$)
s rdf:type o .	o rdfs:subClassOf o .
s p o .	p rdfs:subPropertyOf p .

Figure 7: Schema-level entailment from a single instance-level triple.

Triples	Entailed triple ($\vdash^i_{\text{RDF}}$)
s rdfs:subClassOf $\mathtt{s}_1$., $\mathtt{s}_1$ rdfs:subClassOf $\mathtt{s}_2$.	s rdfs:subClassOf $\mathtt{s}_2$.
p rdfs:subPropertyOf $\mathtt{p}_1$., $\mathtt{p}_1$ rdfs:subPropertyOf $\mathtt{p}_2$.	p rdfs:subPropertyOf $\mathtt{p}_2$.
p rdfs:domain $\mathtt{s}_1$., $\mathtt{s}_1$ rdfs:subClassOf s .	p rdfs:domain s .
p rdfs:range $\mathtt{s}_1$., $\mathtt{s}_1$ rdfs:subClassOf s .	p rdfs:range s .
p rdfs:subPropertyOf $\mathtt{p}_1$., $\mathtt{p}_1$ rdfs:domain s .	p rdfs:domain s .
p rdfs:subPropertyOf $\mathtt{p}_1$., $\mathtt{p}_1$ rdfs:range s .	p rdfs:range s .

Figure 8: Schema-level entailment from two schema triples.

Triples	Entailed triple ($\vdash^i_{\text{RDF}}$)
$\mathtt{s}_1$ rdfs:subClassOf $\mathtt{s}_2$., s rdf:type $\mathtt{s}_1$.	s rdf:type $\mathtt{s}_2$.
$\mathtt{p}_1$ rdfs:subPropertyOf $\mathtt{p}_2$., s $\mathtt{p}_1$ o .	s $\mathtt{p}_2$ o .
p rdfs:domain s ., $\mathtt{s}_1$ p $\mathtt{o}_1$.	$\mathtt{s}_1$ rdf:type s .
p rdfs:range s ., $\mathtt{s}_1$ p $\mathtt{o}_1$.	$\mathtt{o}_1$ rdf:type s .

Figure 9: Instance-level entailment from combining schema and instance triples.

schema-level of db. The other triples, of the forms listed in Figure 2, belong to D, and are called the *instance-level* of db. Observe that S and D provide a way to partition any graph (any triple belongs to exactly one of them).

The saturation of a database db with this aforementioned entailment rule set is denoted $\mathtt{db}^{\propto}$, thus $\mathtt{db}^{\propto} \subseteq \mathtt{db}^{\infty}$.

We consider the BGP queries previously introduced to query the DB fragment of RDF. The *evaluation of a query q against a database* db is exactly the evaluation of q against the graph db, i.e., $q(\mathtt{db})$, and the *answer set of q against* db is $q(\mathtt{db}^{\propto})$, thus $q(\mathtt{db}^{\propto}) \subseteq q(\mathtt{db}^{\infty})$.

In general, user queries may traverse both the schema- and instance-level of the database. In our running example, one can ask for the ranges of the properties describing conference papers, i.e., ClassRelatedToConfPaper(x):- y_1 rdf:type confP ., y_1 y_2 y_3 ., y_2 rdfs:range x .

In some settings, however, the separation between data and schema, which corresponds to many users' intuitive comprehension of the database, may lead them to specify that their queries be evaluated *only against the instance-level* or *only against the schema-level* database.

From a database (DB) perspective, queries whose evaluation is asked against the (saturated) *instance-level database only* are the most familiar. While schema triples are not part of the answer to such an instance-level query, they do impact its answer, because the saturation of the instance-level database (necessary in order to return complete answers) relies on the schema-level triples. E.g., an instance-level query returning the city of WWW2012 is: WWWCity(y):- x name "WWW2012" ., x city y . Instance-level queries can also return classes and properties associated to specific values. For instance, one can ask for the classes to which a given value v belongs, i.e., ClassFinding(x):- v rdf:type x .

From a knowledge representation (KR) perspective, a class of interesting queries can be evaluated over the schema-level database alone. Such queries offer a convenient means to explore the relationships between the classes and properties of a schema, including the implied relationships. For instance, one can ask whether a given class is a subclass of another, e.g., SubclassChecking():- posterCP rdfs:subClassOf paper .; or, what are the classes typing the domain of a given property, e.g., DomainFinding(x):- inProceedingsOf rdfs:domain x . Another schema exploration query can be Schema(x, y, z):- x y z ., returning all triples of the database. By restricting the query to only the schema-level database, the user retrieves all classes, properties, and direct or entailed relationships among them.

In conclusion, our setting is general enough to integrate both DB-style instance-level querying and KR-style schema-level querying, while also allowing a smooth integration of both levels through queries on both database components.

5 Query answering in databases

We investigate two query answering techniques against RDF databases: *saturation-* and *reformulation-based*. Each technique performs a specific *preprocessing* step, either on the database or on the queries, to deal with entailed triples; after which query answering is reduced to query evaluation.

Saturation-based query answering is rather straightforward, since the answer set of a query is computed exactly as it is formally defined. The saturation of the database is computed (using the allowed entailment rules), so that the answer set of every query against the (original) database is obtained by query evaluation against the saturation. The advantage of this approach is that it is easy to implement. Its disadvantages are that database saturation needs time to be computed and space to store all the entailed triples; moreover, the saturation must be somehow recomputed upon every database update.

Reformulation-based query answering reformulates a query q w.r.t. a database $\mathtt{db}$ into another query q' (using the immediate entailment rules), so that the evaluation of q' against the (original) database $\mathtt{db}$, denoted $q'(\mathtt{db})$, is exactly the answer set of q against $\mathtt{db}$ (i.e., $q(\mathtt{db}^{\infty})$). The advantage of reformulation is that the database saturation does not need to be (re)computed. The disadvantage is that every incoming query must be reformulated, which often results in a more

complex query.

We focus on saturation- and reformulation-based query answering only for *instance-level queries*. The following theorem shows that to answer such queries, among our DB fragment's rules shown in Figures 6–9, it suffices to consider only the entailment rules in Figure 9:

Theorem 1. *Let* db *be a database,* t_1 *be a triple of the form* s rdf:type o ., *and* t_2 *be a triple of the form* s p o . $t_1 \in \mathtt{db}^\infty$ *(respectively,* $t_2 \in \mathtt{db}^\infty$*) iff there exists a sequence of application of the rules in Figure 9 leading from* db *to* t_1 *(respectively* t_2*), assuming that each entailment step relies on* db *and all triples previously entailed.*

Proof of Theorem 1. For one direction ($\Leftarrow$), the proof is trivial as the rules of Figure 9 are among those defining $\mathtt{db}^\infty$.

For the converse direction ($\Rightarrow$), let us call a *derivation* of t any sequence of immediate entailment rules that produces the entailed triple t, starting from db. Let us consider, without loss of generality, a *minimal* derivation (i.e., in which removing a step of rule application does not allow deriving t anymore). A derivation can be minimized by gradually removing steps producing entailed triples that are not further reused in the entailment sequence of t. We show for such a *minimal* derivation of an entailed triple t that any step using a rule that is not in Figure 9 can be replaced by a sequence of steps using only rules from Figure 9, leading to another derivation of t. Applying exhaustively the above replacement on the minimization of obtained derivations obviously leads to a derivation of t using the rules in Figure 9 only.

Consider a minimal derivation of t using the immediate entailment rule from Figure 9: s rdfs:subClassOf o .$, \mathtt{s}_1$ rdf:type s . $\vdash_{\text{RDF}}$ $\mathtt{s}_1$ rdf:type o . While the triple $\mathtt{s}_1$ rdf:type s . is either in db or produced by a rule from Figure 9 (only the rules in Figure 9 produce such a triple), the triple s rdfs:subClassOf o . may result from the triples {s rdfs:subClassOf $\mathtt{o}_n$.$, \mathtt{o}_n$ rdfs:subClassOf $\mathtt{o}_{n-1}$.$, \ldots,$ $\mathtt{o}_1$ rdfs:subClassOf o .} $\subseteq$ db and n applications of the rule s rdfs:subClassOf o ., o rdfs:subClassOf $\mathtt{o}_1$. $\vdash_{\text{RDF}}$ s rdfs:subClassOf $\mathtt{o}_1$. from Figure 8 (only that rule produces triples of the form s rdfs:subClassOf o .). Observe that we do not have to consider the rules from Figure 6 in a *minimal* derivation. It is therefore easy to see that the application of s rdfs:subClassOf o .$, \mathtt{s}_1$ rdf:type s . $\vdash_{\text{RDF}}$ $\mathtt{s}_1$ rdf:type o . in the derivation of t can be replaced by the following sequence:
s rdfs:subClassOf $\mathtt{o}_n$.$, \mathtt{s}_1$ rdf:type s . $\vdash_{\text{RDF}}$ $\mathtt{s}_1$ rdf:type $\mathtt{o}_n$.,
$\mathtt{o}_n$ rdfs:subClassOf $\mathtt{o}_{n-1}$.$, \mathtt{s}_1$ rdf:type $\mathtt{o}_n$. $\vdash_{\text{RDF}}$
$\mathtt{s}_1$ rdf:type $\mathtt{o}_{n-1}$.,
$\ldots$,
$\mathtt{o}_1$ rdfs:subClassOf o .$, \mathtt{s}_1$ rdf:type $\mathtt{o}_1$. $\vdash_{\text{RDF}}$ $\mathtt{s}_1$ rdf:type o .

The rest of the proof is omitted as it amounts to show, similarly as above, that the claim also holds for the three other immediate entailment rules of Figure 9. □

6 Saturation-based query answering

The first query answering technique is based on our `Saturate` algorithm, which computes the instance-level saturation of a given database. Evaluating the original query against this saturation then yields the exact answer set.

$$\frac{\{\mathbf{c}_1 \text{ rdfs:subClassOf } \mathbf{c}_2 \,.\, \mathtt{s} \text{ rdf:type } \mathbf{c}_1 \,.\} \subseteq \mathtt{db}}{\mathtt{db} = \mathtt{db} \cup \{\mathtt{s} \text{ rdf:type } \mathbf{c}_2 \,.\}} \quad (1)$$

$$\frac{\{\mathbf{p} \text{ rdfs:domain } \mathbf{c} \,.\, \mathtt{s}\ \mathbf{p}\ \mathtt{o} \,.\} \subseteq \mathtt{db}}{\mathtt{db} = \mathtt{db} \cup \{\mathtt{s} \text{ rdf:type } \mathbf{c} \,.\}} \quad (2)$$

$$\frac{\{\mathbf{p} \text{ rdfs:range } \mathbf{c} \,.\, \mathtt{s}\ \mathbf{p}\ \mathtt{o} \,.\} \subseteq \mathtt{db}}{\mathtt{db} = \mathtt{db} \cup \{\mathtt{o} \text{ rdf:type } \mathbf{c} \,.\}} \quad (3)$$

$$\frac{\{\mathbf{p}_1 \text{ rdfs:subPropertyOf } \mathbf{p}_2 \,.\, \mathtt{s}\ \mathbf{p}_1\ \mathtt{o} \,.\} \subseteq \mathtt{db}}{\mathtt{db} = \mathtt{db} \cup \{\mathtt{s}\ \mathbf{p}_2\ \mathtt{o} \,.\}} \quad (4)$$

Figure 10: Saturation rules for an RDF database db.

$$\begin{aligned}
&\mathtt{Saturate}^0(\mathtt{db}) = \mathtt{db}\\
&\mathtt{Saturate}^1(\mathtt{db}) = \mathtt{Saturate}^0(\mathtt{db}) \cup\\
&\qquad\qquad \{\text{doi}_1 \text{ rdf:type confP .},\\
&\qquad\qquad\ \text{doi}_1 \text{ rdf:type paper .},\\
&\qquad\qquad\ \text{doi}_1 \text{ hasAuthor } _{:}b_1 \text{ .},\\
&\qquad\qquad\ _{:}b_2 \text{ rdf:type conference .}\}\\
&\mathtt{Saturate}^2(\mathtt{db}) = \mathtt{Saturate}^1(\mathtt{db})
\end{aligned}$$

Figure 11: Sample saturation of a database db.

6.1 Database saturation

Our Saturate Algorithm relies on the saturation rules in Figure 10, which are a direct implementation of the entailment rules in Figure 9. In Figure 10 and in the sequel, the bold symbols (possibly with subscripts) **c** for a class, and **p** for a property, denote some unspecified values.

The rules in Figure 10 define a set of database transformations of the form $\frac{input}{output}$, where both the input and the output are databases. Intuitively, given a database db, Saturate(db) applies exhaustively the rules in Figure 10, on db plus all the gradually generated triples.

The output of Saturate(db) is defined as the fixpoint $\mathtt{Saturate}^\infty(\mathtt{db})$, where:

$$\begin{aligned}
&\mathtt{Saturate}^0(\mathtt{db}) = \mathtt{db}\\
&\mathtt{Saturate}^{k+1}(\mathtt{db}) = \mathtt{Saturate}^k(\mathtt{db}) \cup \{t_3 \mid \exists i \in [1,4] \text{ s.t.}\\
&\qquad \text{applying rule } (i) \text{ on some } \{t_1, t_2\} \subseteq \mathtt{Saturate}^k(\mathtt{db})\\
&\qquad \text{produces } t_3 \text{ with } t_2 \notin \mathtt{Saturate}^{j<k-1}(\mathtt{db})\}
\end{aligned}$$

Example 5 (Continued)**.** *The saturation of* db *is shown in Figure 11.*

Theorem 2 shows that our saturation algorithm terminates and provides an upper bound for the size of its output.

Theorem 2. *For a database* db*, the size (number of triples) of the output of* Saturate(db) *is in* $O(\#\mathtt{db}^2)$*, where* #db *is the size (number of triples) of* db*.*

Proof of Theorem 2. A close examination of the saturation rules exhibits producer-consumer dependencies among rules. For instance, triples produced

by the rule (1) can only be used to further apply the same rule. We say rule (1) can only *feed* itself. One can similarly see that rules (2) and (3) can only feed rule (1), and rule (4) can only feed itself plus rules (2) and (3).

Given the two possible forms of instance-level triples, we write the saturation schemes based on the above dependencies using regular expressions as follows.

Instance-level triple	Saturation scheme
s rdf:type o .	$(1)^*$
s p o .	$(4)^*.((2)+(3)).(1)^*$

This said, we provide now an upper bound for the size of Saturate(db), when db contains the single instance-level triple s rdf:type o . or s p o . Assume db = $\langle$S, D$\rangle$ and let #S and #D the sizes (number of triples) of S and D respectively.

- The triple s rdf:type o . can be transformed at most #S times by the sequence of rules $(1)^*$ (there is at most #S schema-level triples in db).
- The triple s p o . can be transformed at most #S times by the sequence of rules $(4)^*.(2).(1)^*$ and also at most #S times by the sequence of rules $(4)^*.(3).(1)^*$ (there is at most #S schema-level triples in db).

As a result, the worst-case is for a triple s p o .: the overall upper bound for the size of the output of Saturate(db) is $2 * \#\mathtt{S}$.

The above result easily generalizes to a database whose instance-level is of size #D: $2 * \#\mathtt{S} * \#\mathtt{D}$. Given that $\#\mathtt{db} = \#\mathtt{S} + \#\mathtt{D}$, the size of the output of Saturate(q) is in $O(\#\mathtt{db}^2)$. □

6.2 Saturation maintenance upon updates

As mentioned earlier, saturation-based query answering is efficient when the saturation is pre-computed, so that query answering reduces to direct evaluation at query time. However, when database updates are allowed, the saturation must be somehow recomputed upon every update.

We propose here saturation *maintenance* upon two kinds of updates: *insertion* – adding triples to the database and *deletion* – removing triples from the database.

We extend the previous notion of database saturation, so that it becomes a *multiset* in which a triple appears as many times as it can be entailed. Formally, given a database db, the saturation is now defined as the fixpoint $\mathtt{Saturate}_+^\infty(\mathtt{db})$ obtained from the following $\mathtt{Saturate}_+$ algorithm; where $\uplus$ is the union operator for multisets.

$$\mathtt{Saturate}_+^0(\mathtt{db}) = \mathtt{db}$$
$$\mathtt{Saturate}_+^{k+1}(\mathtt{db}) = \mathtt{Saturate}_+^k(\mathtt{db}) \uplus \{t_3 \mid \exists i \in [1,4] \text{ s.t. applying rule } (i) \text{ on some } \{t_1, t_2\} \subseteq \mathtt{Saturate}_+^k(\mathtt{db}) \text{ produces } t_3 \text{ with } t_2 \notin \mathtt{Saturate}_+^{j<k}(\mathtt{db})\}$$

The following Property exhibits the obvious relationship between the set-based saturation and the multiset-based saturation of a database; set is the function giving the set of elements occurring in a given multiset.

Property 1. Saturate(db) = set(Saturate$_+$(db)) *holds for any RDF database* db.

$$
\begin{aligned}
\texttt{Saturate}^0_+(\texttt{db}) &= \texttt{db}\\
\texttt{Saturate}^1_+(\texttt{db}) &= \texttt{Saturate}^0_+(\texttt{db}) \uplus \\
&\quad \{\text{doi}_1 \text{ rdf:type confP .},\\
&\quad\ \text{doi}_1 \text{ rdf:type paper .},\\
&\quad\ \text{doi}_1 \text{ rdf:type paper .},\\
&\quad\ \text{doi}_1 \text{ hasAuthor } \_{:}b_1 \text{ .},\\
&\quad\ \text{doi}_1 \text{ rdf:type confP .},\\
&\quad\ \_{:}b_2 \text{ rdf:type conference .},\\
&\quad\ \_{:}b_2 \text{ rdf:type conference .}\}\\
\texttt{Saturate}^2_+(\texttt{db}) &= \texttt{Saturate}^1_+(\texttt{db}) \uplus \\
&\quad \{\text{doi}_1 \text{ rdf:type paper .}\\
&\quad\ \text{doi}_1 \text{ rdf:type paper .},\\
&\quad\ \text{doi}_1 \text{ rdf:type paper .}\}\\
\texttt{Saturate}^3_+(\texttt{db}) &= \texttt{Saturate}^2_+(\texttt{db})
\end{aligned}
$$

Figure 12: Sample multiset-based saturation.

Example 6 (Continued)**.** *Consider again the previously introduced database* db*. Its multiset-based saturation is shown in Figure 12.*

With the multiset-based saturation and Property 1 in place, Theorem 3 shows how saturation can be maintained upon update; $\setminus_+$ is the difference operator for multisets.

Theorem 3. *Let* $\texttt{db} = \langle \texttt{S}, \texttt{D} \rangle$ *be a database.*
Insertion: $\texttt{Saturate}_+(\texttt{db} \cup \{t\}) =$

- $\texttt{Saturate}_+(\texttt{db})$ *if* $t \in \texttt{db}$*, otherwise,* $t \notin \texttt{db}$ *and*
- $\texttt{Saturate}_+(\texttt{db}) \uplus [\texttt{Saturate}_+(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}]$ *if* t *is an instance-level triple;*
- $\texttt{Saturate}_+(\texttt{db}) \uplus \{t\} \uplus \biguplus_{t' \in \texttt{D}'} [\texttt{Saturate}_+(\langle \texttt{S}, \{t'\}\rangle) \setminus_+ \texttt{S}]$*, where the multiset* $\texttt{D}'$ *is* $\{t_3 \mid \exists i \in [1,4]$ *such that applying rule (i) on* $\{t, t_2\}$ *with* $t_2 \in \texttt{Saturate}_+(\texttt{db})$ *yields* $t_3\}$*, if* t *is a schema-level triple.*

Deletion: $\texttt{Saturate}_+(\texttt{db} \setminus \{t\}) =$

- $\texttt{Saturate}_+(\texttt{db})$ *if* $t \notin \texttt{db}$*, otherwise,* $t \in \texttt{db}$ *and*
- $\texttt{Saturate}_+(\texttt{db}) \setminus_+ [\texttt{Saturate}_+(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}]$ *if* t *is an instance-level triple;*
- $(\texttt{Saturate}_+(\texttt{db}) \setminus_+ \{t\}) \setminus_+ (\biguplus_{t' \in \texttt{D}'} [\texttt{Saturate}_+(\langle \texttt{S}, \{t'\}\rangle) \setminus_+ \texttt{S}])$*, where the multiset* $\texttt{D}'$ *is* $\{t_3 \mid \exists i \in [1,4]$ *such that applying rule (i) on* $\{t, t_2\}$ *with* $t_2 \in \texttt{Saturate}_+(\texttt{db})$ *yields* $t_3\}$*, if* t *is a schema-level triple.*

Proof of Theorem 3. Let us consider the three cases for insertions.

1. db is a set of triples and $t \in \texttt{db}$, so we have $\texttt{db} \cup \{t\} = \texttt{db}$, thus $\texttt{Saturate}_+(\texttt{db} \cup \{t\}) = \texttt{Saturate}_+(\texttt{db})$.

2. Given that $t \notin \texttt{db}$ and t is an instance-level triple,
$\texttt{Saturate}_+(\texttt{db} \cup \{t\}) = \texttt{Saturate}_+(\texttt{db}) \uplus [\texttt{Saturate}_+(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}]$
is proved by showing the more general result:
$\texttt{Saturate}_+(\langle \texttt{S}, \texttt{D}_1 \uplus \texttt{D}_2\rangle) = \texttt{Saturate}_+(\langle \texttt{S}, \texttt{D}_1\rangle) \uplus [\texttt{Saturate}_+(\langle \texttt{S}, \texttt{D}_2\rangle) \setminus_+ \texttt{S}]$.
Indeed, observe that $\texttt{Saturate}_+(\texttt{db} \cup \{t\}) = \texttt{Saturate}_+(\langle \texttt{S}, \texttt{D} \uplus \{t\}\rangle)$, since

$t \notin \mathtt{db}$. The proof is by induction on the number k of saturation steps: $\mathtt{Saturate}_+^k(\langle \mathtt{S}, \mathtt{D}_1 \uplus \mathtt{D}_2\rangle) = \mathtt{Saturate}_+^k(\langle \mathtt{S}, \mathtt{D}_1\rangle) \uplus [\mathtt{Saturate}_+^k(\langle \mathtt{S}, \mathtt{D}_2\rangle) \setminus_+ \mathtt{S}]$.

Base step: By definition $\mathtt{Saturate}_+^0(\langle \mathtt{S}, \mathtt{D}_1 \uplus \mathtt{D}_2\rangle) = \langle \mathtt{S}, \mathtt{D}_1 \uplus \mathtt{D}_2\rangle$ and $\mathtt{Saturate}_+^0(\langle \mathtt{S}, \mathtt{D}_1\rangle) \uplus [\mathtt{Saturate}_+^0(\langle \mathtt{S}, \mathtt{D}_2\rangle) \setminus_+ \mathtt{S}] = \langle \mathtt{S}, \mathtt{D}_1\rangle \uplus [\langle \mathtt{S}, \mathtt{D}_2\rangle \setminus_+ \mathtt{S}] = \langle \mathtt{S}, \mathtt{D}_1 \uplus \mathtt{D}_2\rangle$. Thus, $\mathtt{Saturate}_+^0(\langle \mathtt{S}, \mathtt{D}_1 \uplus \mathtt{D}_2\rangle) = \mathtt{Saturate}_+^0(\langle \mathtt{S}, \mathtt{D}_1\rangle) \uplus [\mathtt{Saturate}_+^0(\langle \mathtt{S}, \ \rangle) \setminus_+ \mathtt{S}]$ holds.

Inductive step: Suppose that $\mathtt{Saturate}_+^k(\langle \mathtt{S}, \mathtt{D}_1 \uplus \mathtt{D}_2\rangle) = \mathtt{Saturate}_+^k(\langle \mathtt{S}, \mathtt{D}_1\rangle) \uplus [\mathtt{Saturate}_+^k(\langle \mathtt{S}, \mathtt{D}_2\rangle) \setminus_+ \mathtt{S}]$ for $k < \alpha$ and let us show that it still holds for $k = \alpha$.

By definition, $\mathtt{Saturate}_+^\alpha(\langle \mathtt{S}, \mathtt{D}_1\rangle) \uplus [\mathtt{Saturate}_+^\alpha(\langle \mathtt{S}, \mathtt{D}_2\rangle) \setminus_+ \mathtt{S}] = [\mathtt{Saturate}_+^{\alpha-1}(\langle \mathtt{S}, \mathtt{D}_1\rangle) \uplus \{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \mathtt{Saturate}_+^{\alpha-1}(\langle \mathtt{S}, \mathtt{D}_1\rangle)$ yields t_3 with $t_2 \notin \mathtt{Saturate}_+^{j<\alpha-1}(\langle \mathtt{S}, \mathtt{D}_1\rangle)\}] \uplus [(\mathtt{Saturate}_+^{\alpha-1}(\langle \mathtt{S}, \mathtt{D}_2\rangle) \uplus \{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \mathtt{Saturate}_+^{\alpha-1}(\langle \mathtt{S}, \mathtt{D}_2\rangle)$ yields t_3 with $t_2 \notin \mathtt{Saturate}_+^{j<\alpha-1}(\langle \mathtt{S}, \mathtt{D}_2\rangle)\}) \setminus_+ \mathtt{S}]$ holds.

By the semantics of the $\uplus$ and $\setminus_+$ operators, $\mathtt{Saturate}_+^\alpha(\langle \mathtt{S}, \mathtt{D}_1\rangle) \uplus [\mathtt{Saturate}_+^\alpha(\langle \mathtt{S}, \mathtt{D}_2\rangle) \setminus_+ \mathtt{S}] = [\mathtt{Saturate}_+^{\alpha-1}(\langle \mathtt{S}, \mathtt{D}_1\rangle) \uplus (\mathtt{Saturate}_+^{\alpha-1}(\langle \mathtt{S}, \mathtt{D}_2\rangle) \setminus_+ \mathtt{S})] \uplus \{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \mathtt{Saturate}_+^{\alpha-1}(\langle \mathtt{S}, \mathtt{D}_1\rangle) \uplus [\mathtt{Saturate}_+^{\alpha-1}(\langle \mathtt{S}, \mathtt{D}_2\rangle) \setminus_+ \mathtt{S}]$ yields t_3 with $t_2 \notin \mathtt{Saturate}_+^{j<\alpha-1}(\langle \mathtt{S}, \mathtt{D}_1\rangle) \uplus [\mathtt{Saturate}_+^{j<\alpha-1}(\langle \mathtt{S}, \mathtt{D}_2\rangle) \setminus_+ \mathtt{S}]\}$ holds.

By IH, $\mathtt{Saturate}_+^\alpha(\langle \mathtt{S}, \mathtt{D}_1\rangle) \uplus [\mathtt{Saturate}_+^\alpha(\langle \mathtt{S}, \mathtt{D}_2\rangle) \setminus_+ \mathtt{S}] = \mathtt{Saturate}_+^{\alpha-1}(\langle \mathtt{S}, \mathtt{D}_1 \uplus \mathtt{D}_2\rangle) \uplus \{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \mathtt{Saturate}_+^{\alpha-1}(\langle \mathtt{S}, \mathtt{D}_1 \uplus \mathtt{D}_2\rangle)$ yields t_3 with $t_2 \notin \mathtt{Saturate}_+^{j<\alpha-1}(\langle \mathtt{S}, \mathtt{D}_1 \uplus \mathtt{D}_2\rangle)\}$ holds. Therefore, $\mathtt{Saturate}_+^\alpha(\langle \mathtt{S}, \mathtt{D}_1\rangle) \uplus [\mathtt{Saturate}_+^\alpha(\langle \mathtt{S}, \mathtt{D}_2\rangle) \setminus_+ \mathtt{S}] = \mathtt{Saturate}_+^\alpha(\langle \mathtt{S}, \mathtt{D}_1 \uplus \mathtt{D}_2\rangle)$ holds.

3. Given that $t \notin \mathtt{db}$ and t is a schema-level triple, we prove that $\mathtt{Saturate}_+(\mathtt{db} \cup \{t\}) = \mathtt{Saturate}_+(\mathtt{db}) \uplus \{t\} \uplus \biguplus_{t' \in \mathtt{D}'} [\mathtt{Saturate}_+(\langle \mathtt{S}, \{t'\}\rangle) \setminus_+ \mathtt{S}]$, where the multiset $\mathtt{D}'$ is $\{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on $\{t, t_2\}$ with $t_2 \in \mathtt{Saturate}_+(\mathtt{db})$ yields $t_3\}$. We actually show this by induction on the number k of saturation steps: $\mathtt{Saturate}_+^k(\mathtt{db} \cup \{t\}) = \mathtt{Saturate}_+^k(\mathtt{db}) \uplus \{t\} \uplus \biguplus_{l=0}^k \biguplus_{t'_l \in \mathtt{D}'_l} [\mathtt{Saturate}_+^{k-l}(\langle \mathtt{S}, \{t'_l\}\rangle) \setminus_+ \mathtt{S}]$, where the multiset $\mathtt{D}'_l$ is $\emptyset$ for $l = 0$ and $\{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on $\{t, t_2\}$ yields t_3 with $t_2 \in \mathtt{Saturate}_+^{l-1}(\mathtt{db})$ and $t_2 \notin \mathtt{Saturate}_+^{j<l-1}(\mathtt{db})\}$ for $l > 0$. Indeed, observe that, by definition, $\mathtt{D}' = \biguplus_{l=0}^\infty \mathtt{D}'_l$, i.e., whenever the saturation fixpoint is reached.

Base step: By definition, $\mathtt{Saturate}_+^0(\mathtt{db} \cup \{t\}) = \mathtt{db} \cup \{t\}$ holds. In turn, by definition, $\mathtt{Saturate}_+^0(\mathtt{db}) \uplus \{t\} \uplus \biguplus_{l=0}^0 \biguplus_{t'_l \in \mathtt{D}'_l} [\mathtt{Saturate}_+^{0-l}(\langle \mathtt{S}, \mathtt{D}'_l\rangle) \setminus_+ \mathtt{S}] = \mathtt{db} \uplus \{t\} = \mathtt{db} \cup \{t\}$ since $t \notin \mathtt{db}$. Therefore, $\mathtt{Saturate}_+^0(\mathtt{db} \cup \{t\}) = \mathtt{Saturate}_+^0(\mathtt{db}) \uplus \{t\} \uplus \biguplus_{l=0}^0 \biguplus_{t'_l \in \mathtt{D}'_l} [\mathtt{Saturate}_+^{0-l}(\langle \mathtt{S}, \mathtt{D}'_l\rangle) \setminus_+ \mathtt{S}]$ holds.

Inductive step: Suppose that $\mathtt{Saturate}_+^k(\mathtt{db} \cup \{t\}) = \mathtt{Saturate}_+^k(\mathtt{db}) \uplus \{t\} \uplus \biguplus_{l=0}^k \biguplus_{t'_l \in \mathtt{D}'_l} [\mathtt{Saturate}_+^{k-l}(\langle \mathtt{S}, \{t'_l\}\rangle) \setminus_+ \mathtt{S}]$ holds for $k < \alpha$ and let us show that it still holds for $k = \alpha$.

By definition, $\mathtt{Saturate}_+^\alpha(\mathtt{db}) \uplus \{t\} \uplus \biguplus_{l=0}^\alpha \biguplus_{t'_l \in \mathtt{D}'_l} [\mathtt{Saturate}_+^{\alpha-l}(\langle \mathtt{S}, \{t'_l\}\rangle) \setminus_+ \mathtt{S}] = (\mathtt{Saturate}_+^{\alpha-1}(\mathtt{db}) \uplus \{t_3 \mid \exists i \in [1,4]$ such that

applying rule (i) on some $\{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1}(\texttt{db})$ yields t_3 with $t_2 \notin \texttt{Saturate}_+^{j<\alpha-1}(\texttt{db})\}) \uplus \{t\} \uplus \biguplus_{t'_\alpha \in \texttt{D}'_\alpha}[\texttt{Saturate}_+^0(\langle \texttt{S}, \{t'_\alpha\}\rangle) \setminus_+ \texttt{S}] \uplus \biguplus_{l=0}^{\alpha-1} \biguplus_{t'_l \in \texttt{D}'_l}[(\texttt{Saturate}_+^{\alpha-1-l}(\langle \texttt{S}, \{t'_l\}\rangle) \uplus \{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1-l}(\langle \texttt{S}, \{t'_l\}\rangle)$ yields t_3 with $t_2 \notin \texttt{Saturate}_+^{j<\alpha-1-l}(\langle \texttt{S}, \{t'_l\}\rangle)\}) \setminus_+ \texttt{S}]$ holds.

That is, by the semantics of the $\uplus$ and $\setminus_+$ operators, $\texttt{Saturate}_+^{\alpha}(\texttt{db}) \uplus \{t\} \uplus \biguplus_{l=0}^{\alpha} \biguplus_{t'_l \in \texttt{D}'_l}[\texttt{Saturate}_+^{\alpha-l}(\langle \texttt{S}, \{t'_l\}\rangle) \setminus_+ \texttt{S}] = \texttt{Saturate}_+^{\alpha-1}(\texttt{db}) \uplus \{t\} \uplus \biguplus_{l=0}^{\alpha-1} \biguplus_{t'_l \in \texttt{D}'_l}[\texttt{Saturate}_+^{\alpha-1-l}(\langle \texttt{S}, \{t'_l\}\rangle) \setminus_+ \texttt{S}] \uplus \{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1}(\texttt{db})$ yields t_3 with $t_2 \notin \texttt{Saturate}_+^{j<\alpha-1}(\texttt{db})\} \uplus \biguplus_{t'_\alpha \in \texttt{D}'_\alpha}[\langle \texttt{S}, \{t'_\alpha\}\rangle \setminus_+ \texttt{S}] \uplus \biguplus_{l=0}^{\alpha-1} \biguplus_{t'_l \in \texttt{D}'_l}\{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1-l}(\langle \texttt{S}, \{t'_l\}\rangle)$ yields t_3 with $t_2 \notin \texttt{Saturate}_+^{j<\alpha-1-l}(\langle \texttt{S}, \{t'_l\}\rangle)\}$ holds.

By IH, $\texttt{Saturate}_+^{\alpha}(\texttt{db}) \uplus \{t\} \uplus \biguplus_{l=0}^{\alpha} \biguplus_{t'_l \in \texttt{D}'_l}[\texttt{Saturate}_+^{\alpha-l}(\langle \texttt{S}, \{t'_l\}\rangle) \setminus_+ \texttt{S}] = \texttt{Saturate}_+^{\alpha-1}(\texttt{db} \cup \{t\}) \uplus \{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1}(\texttt{db})$ yields t_3 with $t_2 \notin \texttt{Saturate}_+^{j<\alpha-1}(\texttt{db})\} \uplus \texttt{D}'_\alpha \uplus \biguplus_{l=0}^{\alpha-1} \biguplus_{t'_l \in \texttt{D}'_l}\{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1-l}(\langle \texttt{S}, \{t'_l\}\rangle)$ yields t_3 with $t_2 \notin \texttt{Saturate}_+^{j<\alpha-1-l}(\langle \texttt{S}, \{t'_l\}\rangle)\}$ holds.

By definition of $\texttt{D}'_{0 \leq i \leq \alpha}$, $\texttt{Saturate}_+^{\alpha}(\texttt{db}) \uplus \{t\} \uplus \biguplus_{l=0}^{\alpha} \biguplus_{t'_l \in \texttt{D}'_l}[\texttt{Saturate}_+^{\alpha-l}(\langle \texttt{S}, \{t'_l\}\rangle) \setminus_+ \texttt{S}] = \texttt{Saturate}_+^{\alpha-1}(\texttt{db} \cup \{t\}) \uplus \{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1}(\texttt{db} \cup \{t\})$ yields t_3 with $t_2 \notin \texttt{Saturate}_+^{j<\alpha-1}(\texttt{db} \cup \{t\})\}$ holds. Therefore, $\texttt{Saturate}_+^{\alpha}(\texttt{db}) \uplus \{t\} \uplus \biguplus_{l=0}^{\alpha} \biguplus_{t'_l \in \texttt{D}'_l}[\texttt{Saturate}_+^{\alpha-l}(\langle \texttt{S}, \{t'_l\}\rangle) \setminus_+ \texttt{S}] = \texttt{Saturate}_+^{\alpha}(\texttt{db} \cup \{t\})$ holds.

Let us now consider the three cases for deletions.

1. db is a set of triples and $t \notin \texttt{db}$, so we have $\texttt{db} \setminus \{t\} = \texttt{db}$, thus $\texttt{Saturate}_+(\texttt{db} \setminus \{t\}) = \texttt{Saturate}_+(\texttt{db})$.

2. Given that $t \in \texttt{db}$ and t is an instance-level triple, $\texttt{Saturate}_+(\texttt{db} \setminus \{t\}) = \texttt{Saturate}_+(\texttt{db}) \setminus_+ [\texttt{Saturate}_+(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}]$ is shown on the number k of saturation steps: $\texttt{Saturate}_+^k(\texttt{db} \setminus \{t\}) = \texttt{Saturate}_+^k(\texttt{db}) \setminus_+ [\texttt{Saturate}_+^k(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}]$.

   Base step: By definition $\texttt{Saturate}_+^0(\texttt{db} \setminus \{t\}) = \texttt{db} \setminus \{t\}$ and $\texttt{Saturate}_+^0(\texttt{db}) \setminus_+ [\texttt{Saturate}_+^0(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}] = \texttt{db} \setminus_+ [\langle \texttt{S}, \{t\}\rangle \setminus_+ \texttt{S}] = \texttt{db} \setminus_+ \{t\} = \texttt{db} \setminus \{t\}$ since $t \in \texttt{db}$, thus $\texttt{Saturate}_+^0(\texttt{db} \setminus \{t\}) = \texttt{Saturate}_+^0(\texttt{db}) \setminus_+ [\texttt{Saturate}_+^0(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}]$ holds.

   Inductive step: Suppose that $\texttt{Saturate}_+^k(\texttt{db} \setminus \{t\}) = \texttt{Saturate}_+^k(\texttt{db}) \setminus_+ [\texttt{Saturate}_+^k(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}]$ for $k < \alpha$ and let us show that it still holds for $k = \alpha$.

   By definition, $\texttt{Saturate}_+^{\alpha}(\texttt{db}) \setminus_+ [\texttt{Saturate}_+^{\alpha}(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}] = (\texttt{Saturate}_+^{\alpha-1}(\texttt{db}) \uplus \{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1}(\texttt{db})$ yields t_3 with $t_2 \notin \texttt{Saturate}_+^{j<\alpha-1}(\texttt{db})\}) \setminus_+ [(\texttt{Saturate}_+^{\alpha-1}(\langle \texttt{S}, \{t\}\rangle) \uplus \{t_3 \mid \exists i \in [1,4]$ such that applying

rule (i) on some $\{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1}(\langle \texttt{S}, \{t\}\rangle)$ yields t_3 with $t_2 \notin \texttt{Saturate}_+^{j<\alpha-1}(\langle \texttt{S}, \{t\}\rangle)\}) \setminus_+ \texttt{S}]$ holds.

By the semantics of the $\uplus$ and $\setminus_+$ operators, and since $\texttt{Saturate}_+^{\alpha-1}(\langle \texttt{S}, \{t\}\rangle) \subseteq \texttt{Saturate}_+^{\alpha}(\texttt{db})$, $\texttt{Saturate}_+^{\alpha}(\texttt{db}) \setminus_+ [\texttt{Saturate}_+^{\alpha}(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}] = (\texttt{Saturate}_+^{\alpha-1}(\texttt{db}) \setminus_+ [\texttt{Saturate}_+^{\alpha-1}(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}]) \uplus (\{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1}(\texttt{db})$ yields t_3 with $t_2 \notin \texttt{Saturate}_+^{j<\alpha-1}(\texttt{db})\} \setminus_+ \{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1}(\langle \texttt{S}, \{t\}\rangle)$ yields t_3 with $t_2 \notin \texttt{Saturate}_+^{j<\alpha-1}(\langle \texttt{S}, \{t\}\rangle)\})$ holds.

By IH, $\texttt{Saturate}_+^{\alpha}(\texttt{db}) \setminus_+ [\texttt{Saturate}_+^{\alpha}(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}] = \texttt{Saturate}_+^{\alpha-1}(\texttt{db} \setminus \{t\}) \uplus (\{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1}(\texttt{db})$ yields t_3 with $t_2 \notin \texttt{Saturate}_+^{j<\alpha-1}(\texttt{db})\} \setminus_+ \{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1}(\langle \texttt{S}, \{t\}\rangle)$ yields t_3 with $t_2 \notin \texttt{Saturate}_+^{j<\alpha-1}(\langle \texttt{S}, \{t\}\rangle)\})$ holds.

That is, $\texttt{Saturate}_+^{\alpha}(\texttt{db}) \setminus_+ [\texttt{Saturate}_+^{\alpha}(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}] = \texttt{Saturate}_+^{\alpha-1}(\texttt{db} \setminus \{t\}) \uplus \{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1}(\texttt{db}) \setminus_+ [\texttt{Saturate}_+^{\alpha-1}(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}]$ yields t_3 with $t_2 \notin \texttt{Saturate}_+^{j<\alpha-1}(\texttt{db}) \setminus_+ [\texttt{Saturate}_+^{j<\alpha-1}(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}]\}$ holds.

By IH, $\texttt{Saturate}_+^{\alpha}(\texttt{db}) \setminus_+ [\texttt{Saturate}_+^{\alpha}(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}] = \texttt{Saturate}_+^{\alpha-1}(\texttt{db} \setminus \{t\}) \uplus \{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1}(\texttt{db} \setminus \{t\})$ yields t_3 with $t_2 \notin \texttt{Saturate}_+^{j<\alpha-1}(\texttt{db} \setminus \{t\})\}$ holds. Therefore, $\texttt{Saturate}_+^{\alpha}(\texttt{db}) \setminus_+ [\texttt{Saturate}_+^{\alpha}(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}] = \texttt{Saturate}_+^{\alpha}(\texttt{db} \cup \{t\})$ holds.

3. Given that $t \in \texttt{db}$ and t is a schema-level triple, $\texttt{Saturate}_+(\texttt{db} \setminus \{t\}) = (\texttt{Saturate}_+(\texttt{db}) \setminus_+ \{t\}) \setminus_+ (\biguplus_{t' \in \texttt{D}'} [\texttt{Saturate}_+(\langle \texttt{S}, \{t'\}\rangle) \setminus_+ \texttt{S}])$, where the multiset $\texttt{D}'$ is $\{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on $\{t, t_2\}$ with $t_2 \in \texttt{Saturate}_+(\texttt{db})$ yields $t_3\}$. We actually show this by induction on the number k of saturation steps: $\texttt{Saturate}_+^k(\texttt{db} \setminus \{t\}) = (\texttt{Saturate}_+^k(\texttt{db}) \setminus_+ \{t\}) \setminus_+ (\biguplus_{l=0}^{k} \biguplus_{t'_l \in \texttt{D}'_l} [\texttt{Saturate}_+^{k-l}(\langle \texttt{S}, \{t'_l\}\rangle) \setminus_+ \texttt{S}])$, where the multiset $\texttt{D}'_l$ is $\emptyset$ for $l = 0$ and $\{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on $\{t, t_2\}$ yields t_3 with $t_2 \in \texttt{Saturate}_+^{l-1}(\texttt{db})$ and $t_2 \notin \texttt{Saturate}_+^{j<l-1}(\texttt{db})\}$ for $l > 0$. Indeed, observe that, by definition, $\texttt{D}' = \biguplus_{l=0}^{\infty} \texttt{D}'_l$, i.e., whenever the saturation fixpoint is reached.

Base step: By definition, $\texttt{Saturate}_+^0(\texttt{db} \setminus \{t\}) = \texttt{db} \cup \{t\}$ holds. In turn, by definition, $(\texttt{Saturate}_+^0(\texttt{db}) \setminus_+ \{t\}) \setminus_+ (\biguplus_{l=0}^{0} \biguplus_{t'_l \in \texttt{D}'_l} [\texttt{Saturate}_+^{0-l}(\langle \texttt{S}, \texttt{D}'_l\rangle) \setminus_+ \texttt{S}]) = \texttt{db} \setminus_+ \{t\} = \texttt{db} \setminus \{t\}$ since $t \notin \texttt{db}$. Therefore, $\texttt{Saturate}_+^0(\texttt{db} \setminus \{t\}) = (\texttt{Saturate}_+^0(\texttt{db}) \setminus_+ \{t\}) \setminus_+ (\biguplus_{l=0}^{0} \biguplus_{t'_l \in \texttt{D}'_l} [\texttt{Saturate}_+^{0-l}(\langle \texttt{S}, \texttt{D}'_l\rangle) \setminus_+ \texttt{S}])$ holds.

Inductive step: Suppose that $\texttt{Saturate}_+^k(\texttt{db} \setminus \{t\}) = (\texttt{Saturate}_+^k(\texttt{db}) \setminus_+ \{t\}) \setminus_+ (\biguplus_{l=0}^{k} \biguplus_{t'_l \in \texttt{D}'_l} [\texttt{Saturate}_+^{k-l}(\langle \texttt{S}, \{t'_l\}\rangle) \setminus_+ \texttt{S}])$ holds for $k < \alpha$ and let us show that it still holds for $k = \alpha$.

By definition, $(\texttt{Saturate}_+^{\alpha}(\texttt{db}) \setminus_+ \{t\}) \setminus_+ (\biguplus_{l=0}^{\alpha} \biguplus_{t'_l \in \texttt{D}'_l} [\texttt{Saturate}_+^{\alpha-l}(\langle \texttt{S}, \{t'_l\}\rangle) \setminus_+ \texttt{S}]) = ([\texttt{Saturate}_+^{\alpha-1}(\texttt{db}) \uplus \{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1}(\texttt{db})$ yields t_3 with $t_2 \notin \texttt{Saturate}_+^{j<\alpha-1}(\texttt{db})\}] \setminus_+ \{t\}) \setminus_+ (\biguplus_{t'_\alpha \in \texttt{D}'_\alpha} [\texttt{Saturate}_+^{0}(\langle \texttt{S}, \{t'_\alpha\}\rangle) \setminus_+ \texttt{S}]$

$\uplus \biguplus_{l=0}^{\alpha-1} \biguplus_{t'_l \in \mathtt{D}'_l} [(\mathtt{Saturate}_+^{\alpha-1-l}(\langle \mathtt{S}, \{t'_l\}\rangle) \uplus \{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \mathtt{Saturate}_+^{\alpha-1-l}(\langle \mathtt{S}, \{t'_l\}\rangle)$ yields t_3 with $t_2 \notin \mathtt{Saturate}_+^{j<\alpha-1-l}(\langle \mathtt{S}, \{t'_l\}\rangle)\}) \setminus_+ \mathtt{S}])$ holds.

That is, by the semantics of the $\uplus$ and $\setminus_+$ operators and since $\biguplus_{l=0}^{\alpha-1} \biguplus_{t'_l \in \mathtt{D}'_l} [\mathtt{Saturate}_+^{\alpha-1-l}(\langle \mathtt{S}, \{t'_l\}\rangle) \setminus_+ \mathtt{S}] \subseteq \mathtt{Saturate}_+^{\alpha-1}(\mathtt{db})$ holds due to $t \in \mathtt{db}$, $(\mathtt{Saturate}_+^{\alpha}(\mathtt{db}) \setminus_+ \{t\}) \setminus_+ (\biguplus_{l=0}^{\alpha} \biguplus_{t'_l \in \mathtt{D}'_l} [\mathtt{Saturate}_+^{\alpha-l}(\langle \mathtt{S}, \{t'_l\}\rangle) \setminus_+ \mathtt{S}]) = [(\mathtt{Saturate}_+^{\alpha-1}(\mathtt{db}) \setminus_+ \{t\}) \setminus_+ \biguplus_{l=0}^{\alpha-1} \biguplus_{t'_l \in \mathtt{D}'_l} (\mathtt{Saturate}_+^{\alpha-1-l}(\langle \mathtt{S}, \{t'_l\}\rangle) \setminus_+ \mathtt{S})] \uplus \{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \mathtt{Saturate}_+^{\alpha-1}(\mathtt{db})$ yields t_3 with $t_2 \notin \mathtt{Saturate}_+^{j<\alpha-1}(\mathtt{db})\} \setminus_+ (\biguplus_{t'_\alpha \in \mathtt{D}'_\alpha} [\mathtt{Saturate}_+^{0}(\langle \mathtt{S}, \{t'_\alpha\}\rangle) \setminus_+ \mathtt{S}] \uplus \biguplus_{l=0}^{\alpha-1} \biguplus_{t'_l \in \mathtt{D}'_l} \{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \mathtt{Saturate}_+^{\alpha-1-l}(\langle \mathtt{S}, \{t'_l\}\rangle)$ yields t_3 with $t_2 \notin \mathtt{Saturate}_+^{j<\alpha-1-l}(\langle \mathtt{S}, \{t'_l\}\rangle)\})$ holds.

By IH, $(\mathtt{Saturate}_+^{\alpha}(\mathtt{db}) \setminus_+ \{t\}) \setminus_+ (\biguplus_{l=0}^{\alpha} \biguplus_{t'_l \in \mathtt{D}'_l} [\mathtt{Saturate}_+^{\alpha-l}(\langle \mathtt{S}, \{t'_l\}\rangle) \setminus_+ \mathtt{S}]) = \mathtt{Saturate}_+^{\alpha-1}(\mathtt{db} \setminus \{t\}) \uplus \{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \mathtt{Saturate}_+^{\alpha-1}(\mathtt{db})$ yields t_3 with $t_2 \notin \mathtt{Saturate}_+^{j<\alpha-1}(\mathtt{db})\} \setminus_+ (\mathtt{D}'_\alpha \uplus \biguplus_{l=0}^{\alpha-1} \biguplus_{t'_l \in \mathtt{D}'_l} \{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \mathtt{Saturate}_+^{\alpha-1-l}(\langle \mathtt{S}, \{t'_l\}\rangle)$ yields t_3 with $t_2 \notin \mathtt{Saturate}_+^{j<\alpha-1-l}(\langle \mathtt{S}, \{t'_l\}\rangle)\})$ holds.

By definition of $\mathtt{D}'_{0 \leq i \leq \alpha}$, $(\mathtt{Saturate}_+^{\alpha}(\mathtt{db}) \setminus_+ \{t\}) \setminus_+ (\biguplus_{l=0}^{\alpha} \biguplus_{t'_l \in \mathtt{D}'_l} [\mathtt{Saturate}_+^{\alpha-l}(\langle \mathtt{S}, \{t'_l\}\rangle) \setminus_+ \mathtt{S}]) = \mathtt{Saturate}_+^{\alpha-1}(\mathtt{db} \setminus \{t\}) \uplus \{t_3 \mid \exists i \in [1,4]$ such that applying rule (i) on some $\{t_1, t_2\} \subseteq \mathtt{Saturate}_+^{\alpha-1}(\mathtt{db} \setminus \{t\})$ yields t_3 with $t_2 \notin \mathtt{Saturate}_+^{j<\alpha-1}(\mathtt{db} \setminus \{t\})\}$ holds. Therefore, $(\mathtt{Saturate}_+^{\alpha}(\mathtt{db}) \setminus_+ \{t\}) \setminus_+ (\biguplus_{l=0}^{\alpha} \biguplus_{t'_l \in \mathtt{D}'_l} [\mathtt{Saturate}_+^{\alpha-l}(\langle \mathtt{S}, \{t'_l\}\rangle) \setminus_+ \mathtt{S}]) = \mathtt{Saturate}_+^{\alpha}(\mathtt{db} \setminus \{t\})$ holds.

□

Theorem 3 reads as follows. Inserting a triple already in the database, or deleting a triple that is not in the database, does not affect the current saturation. Otherwise, inserting (deleting) a given instance- or schema-level triple also adds to (removes from) the current saturation any instance-level triple whose derivation uses this given triple.

From a practical viewpoint, the multiset $\mathtt{Saturate}_+(\mathtt{db})$ can be compactly stored as the set $\mathtt{Saturate}(\mathtt{db})$, for which every triple is tagged with (i) a boolean indicating whether it is either in $\mathtt{db}$ or only entailed by $\mathtt{db}$ and (ii) an integer indicating how many times it appears in $\mathtt{Saturate}_+(\mathtt{db})$. That way, we encode in a single lightweight representation $\mathtt{db}$, $\mathtt{Saturate}(\mathtt{db})$, and $\mathtt{Saturate}_+(\mathtt{db})$.

6.3 From database saturation to saturation-based query answering

Based on the above notion of saturation, we state our saturation-based query answering technique:

Theorem 4. *Given a BGP query q and a database* db*, the following holds:*

$$q(\texttt{db}^{\infty}) = q(\texttt{Saturate}(\texttt{db})) = q(\texttt{set}(\texttt{Saturate}_{+}(\texttt{db}))).$$

The proof for Theorem 4 trivially follows from Theorem 1, the definition of our Saturate algorithm, and Property 1. From a practical viewpoint, based on Section 3, saturation-based query answering can be delegated to *any* RDBMS by storing either Saturate(db) or the aforementioned compact representation of $\texttt{Saturate}_{+}(\texttt{db})$ in the Triple table, and then by evaluating queries using relational evaluation.

Example 7 (Continued)**.** *Consider now the query $q(x, y)$:- x* rdf:type *y . asking for all resources and the classes to which they belong. The answer set of q against the previous database* db *is:*

$$\begin{aligned} &q(\texttt{Saturate}(\texttt{db})) = \\ &q(\texttt{set}(\texttt{Saturate}_{+}(\texttt{db}))) = \\ &\qquad \{\langle \text{doi}_1, \_{:}b_0\rangle, \\ &\qquad\ \langle \text{www2012}, \text{conference}\rangle, \\ &\qquad\ \langle \text{doi}_1, \text{confP}\rangle, \\ &\qquad\ \langle \text{doi}_1, \text{paper}\rangle, \\ &\qquad\ \langle \_{:}b_2, \text{conference}\rangle\}. \end{aligned}$$

7 Reformulation-based query answering

The second query answering technique is based on our Reformulate algorithm. Given a query q and a database db, $\texttt{Reformulate}(q, \texttt{db})$ reformulates q into a set of queries, such that the union of the *non-standard evaluations* (see below) of these queries on db produces $q(\texttt{db}^{\infty})$, the answer set of the original query against the database.

7.1 Query reformulation

Our Reformulate algorithm exhaustively applies the set of rules shown in Figure 13, starting from a query q and a database db. Each rule defines a transformation of the form $\frac{input}{output}$, where the input is of the form ⟨logical condition on db, logical condition on q⟩ and the output is a query q'. Each but not both of the conditions in the input may be empty. Intuitively, each rule produces a new query when the rule's input conditions are satisfied, one by the database db, and the other by some query (either the original query q or a query q' produced by a previous application of a rule). The set of all queries produced by applying the rules is the result of the reformulation of q w.r.t. db.

Partially instantiated queries. We first introduce the notion of partially instantiated queries. Let q be a query. A *partially instantiated query* denoted q_σ, is a query $q_\sigma(\bar{x}_\sigma)$:- $(t_1, \ldots, t_\alpha)_\sigma$ where σ assigns a subset of the variables and blank nodes in q to some values (URIs, blank nodes, literals). In a non-standard fashion, some distinguished variables of q_σ can be bound. If $\sigma = \emptyset$, then q_σ coincides with the original (non-instantiated) query q.

The notions of evaluation and answer set of BGP queries provided in Section 2.2 for graphs and in Section 4 for databases generalize to partially

$$\frac{\langle \mathtt{s}\ y\ \mathtt{o}\ .\ \in q_\sigma\rangle}{q_{\sigma\cup\nu=\{y\to\text{rdf:type}\}}} \tag{5}$$

$$\frac{\langle \mathtt{s}_1\ \mathbf{p}\ \mathtt{o}_1\ .\ \in \mathtt{db}, \mathtt{s}\ y\ \mathtt{o}\ .\ \in q_\sigma\rangle}{q_{\sigma\cup\nu=\{y\to\mathbf{p}\}}} \tag{6}$$

$$\frac{\langle \mathtt{s}_1\ \text{rdfs:subPropertyOf}\ \mathbf{p}\ .\ \in \mathtt{db}, \mathtt{s}\ y\ \mathtt{o}\ .\ \in q_\sigma\rangle}{q_{\sigma\cup\nu=\{y\to\mathbf{p}\}}} \tag{7}$$

$$\frac{\langle \mathbf{p}\ \text{rdfs:subPropertyOf}\ \mathtt{o}_1\ .\ \in \mathtt{db}, \mathtt{s}\ y\ \mathtt{o}\ .\ \in q_\sigma\rangle}{q_{\sigma\cup\nu=\{y\to\mathbf{p}\}}} \tag{8}$$

$$\frac{\langle \mathtt{s}_1\ \text{rdf:type}\ \mathbf{c}\ .\ \in \mathtt{db}, \mathtt{s}\ \text{rdf:type}\ z\ .\ \in q_\sigma\rangle}{q_{\sigma\cup\nu=\{z\to\mathbf{c}\}}} \tag{9}$$

$$\frac{\langle \mathtt{s}_1\ \text{rdfs:subClassOf}\ \mathbf{c}\ .\ \in \mathtt{db}, \mathtt{s}\ \text{rdf:type}\ z\ .\ \in q_\sigma\rangle}{q_{\sigma\cup\nu=\{z\to\mathbf{c}\}}} \tag{10}$$

$$\frac{\langle \mathbf{c}\ \text{rdfs:subClassOf}\ \mathtt{o}\ .\ \in \mathtt{db}, \mathtt{s}\ \text{rdf:type}\ z\ .\ \in q_\sigma\rangle}{q_{\sigma\cup\nu=\{z\to\mathbf{c}\}}} \tag{11}$$

$$\frac{\langle \mathtt{s}_1\ \text{rdfs:domain}\ \mathbf{c}\ .\ \in \mathtt{db}, \mathtt{s}\ \text{rdf:type}\ z\ .\ \in q_\sigma\rangle}{q_{\sigma\cup\nu=\{z\to\mathbf{c}\}}} \tag{12}$$

$$\frac{\langle \mathtt{s}_1\ \text{rdfs:range}\ \mathbf{c}\ .\ \in \mathtt{db}, \mathtt{s}\ \text{rdf:type}\ z\ .\ \in q_\sigma\rangle}{q_{\sigma\cup\nu=\{z\to\mathbf{c}\}}} \tag{13}$$

$$\frac{\langle \mathbf{c}_1\ \text{rdfs:subClassOf}\ \mathbf{c}_2\ .\ \in \mathtt{db}, \mathtt{s}\ \text{rdf:type}\ \mathbf{c}_2\ .\ \in q_\sigma\rangle}{q_{\sigma[\mathtt{s}\ \text{rdf:type}\ \mathbf{c}_2\ ./\mathtt{s}\ \text{rdf:type}\ \mathbf{c}_1\ .]}} \tag{14}$$

$$\frac{\langle \mathbf{p}\ \text{rdfs:domain}\ \mathbf{c}\ .\ \in \mathtt{db}, \mathtt{s}\ \text{rdf:type}\ \mathbf{c}\ .\ \in q_\sigma\rangle}{q_{\sigma[\mathtt{s}\ \text{rdf:type}\ \mathbf{c}\ ./\mathtt{s}\ \mathbf{p}\ y\ .]}} \tag{15}$$

$$\frac{\langle \mathbf{p}\ \text{rdfs:range}\ \mathbf{c}\ .\ \in \mathtt{db}, \mathtt{s}\ \text{rdf:type}\ \mathbf{c}\ .\ \in q_\sigma\rangle}{q_{\sigma[\mathtt{s}\ \text{rdf:type}\ \mathbf{c}\ ./y\ \mathbf{p}\ \mathtt{s}\ .]}} \tag{16}$$

$$\frac{\langle \mathbf{p}_1\ \text{rdfs:subPropertyOf}\ \mathbf{p}_2\ .\ \in \mathtt{db}, \mathtt{s}\ \mathbf{p}_2\ \mathtt{o}\ .\ \in q_\sigma\rangle}{q_{\sigma[\mathtt{s}\ \mathbf{p}_1\ \mathtt{o}\ ./\mathtt{s}\ \mathbf{p}_2\ \mathtt{o}\ .]}} \tag{17}$$

Figure 13: Reformulation rules for a partially instantiated query q_σ w.r.t. a database db.

instantiated queries as follows. Given a database db whose set of values (URIs, blank nodes, literals) is Val(db), a query $q_\sigma(\bar{x}_\sigma)$:- $(t_1, \ldots, t_\alpha)_\sigma$ whose set of variables and blank nodes is VarBl(q_σ), the *evaluation* of q_σ against db is: $q_\sigma(\mathtt{db}) = \{(\bar{x}_\sigma)_\mu \mid \mu : \mathtt{VarBl}(q_\sigma) \to \mathtt{Val}(\mathtt{db})$ is a total assignment s.t.

$((t_1, \ldots, t_\alpha)_\sigma)_\mu \subseteq \mathtt{db}\}$.

The *answer set* of q_σ against db is the evaluation of q_σ against $\mathtt{db}^\infty$, denoted $q_\sigma(\mathtt{db}^\infty)$.

Reformulation rules. The rules (5)–(13) reformulate queries by binding one of their variables, either to the built-in property rdf:type or to a class or property name picked in the database. The other rules (14)–(17) replace some query triple with another, based on schema-level triples.

Consider for instance rule (5). The rule says: if a triple of the form s y o ., i.e., having any kind of subject or object, but having a variable in the property position, appears in q_σ, then create the new query $q_{\sigma\cup\nu}$, which binds y to the built-in property rdf:type. Observe that if y was a distinguished variable in q_σ, a head variable in $q_{\sigma\cup\nu}$ will be bound after the rule application. Now consider rule (6) on some query q_σ. If q_σ contains a triple of the same form s y o ., and the database db contains a triple with any **p** in the property position, the rule creates the new query $q_{\sigma\cup\nu}$ where y is bound to **p**. Rules (7) and (8) instantiate query variables appearing in the property position, to values appearing in a rdfs:subPropertyOf statement of db. The intuition is that both the subject and the object of a rdfs:subPropertyOf statements are properties, therefore they can be used to instantiate the property variable y.

Rules (9)–(13) instantiate the variable z in a query triple of the form s rdf:type z . The RDF meta-model specifies that the values of the rdf:type property are classes. Therefore, the rules bind z to db values of which it can be inferred that they are classes, i.e., those appearing in specific positions in schema-level triples. For instance, if $\mathbf{s}_1$ rdf:type **c** . $\in$ db, then **c** is a class and z in rule (9) can be instantiated to **c**. Similarly, the subject and object of a rdfs:subClassOf statements are used in rules (10) and (11). Finally, Rules (14)–(17) use schema triples to replace (denoted *old triple / new triple*) a triple in the input query with a new triple. Rule (14) exploits rdfs:subClassOf statements: if the query q_σ asks for instances of class $\mathbf{c}_2$ and $\mathbf{c}_1$ is a subclass of $\mathbf{c}_2$, then instances of $\mathbf{c}_1$ should also be returned, and this is what the output query of this rule does. The last three rules are similar.

Example 8 (Continued)**.** *Consider the previously introduced database* db *and query* $q(x, y, z)$*:-* x y z *. asking for the triples of* db *(including the entailed ones). We show how some of the above rules can be used to reformulate* q *w.r.t.* db*.*

1. *Using* q *as input for rule (5) produces the query:* $q_{\{y\to\text{rdf:type}\}}$*, i.e.,* $q(x, y, z)$*:-* x rdf:type z .

2. *Using* $q_{\{y\to\text{rdf:type}\}}$ *as input for rule (11) can lead to:* $q_{\{y\to\text{rdf:type},z\to\text{confP}\}}$*, i.e.,* $q(x, \text{rdf:type}, \text{confP})$*:-* x rdf:type confP .

3. *Finally, using* $q_{\{y\to\text{rdf:type},z\to\text{confP}\}}$ *as input for rule (14) can lead to:* $q(x, \text{rdf:type}, \text{confP})$*:-* x rdf:type _:b_0 .

Query reformulation algorithm. For a query q and a database db, the output of $\mathtt{Reformulate}(q, \mathtt{db})$ is defined as the fixpoint $\mathtt{Reformulate}^\infty(q, \mathtt{db})$, where:

$$\mathtt{Reformulate}^0(q, \mathtt{db}) = \{q\}$$
$$\mathtt{Reformulate}^{k+1}(q, \mathtt{db}) = \mathtt{Reformulate}^k(q, \mathtt{db}) \cup \{q''_{\sigma''} \mid \exists i \in [5, \ldots, 17]$$
$$\text{s.t. applying rule } (i) \text{ on db and some query}$$
$$q'_{\sigma'} \in \mathtt{Reformulate}^k(q, \mathtt{db}) \text{ yields the query } q''_{\sigma''}\}$$

Theorem 5 shows that our reformulation algorithm terminates and provides an upper bound for the size of its output.

Theorem 5. *Given a BGP query q and a database* db*, the size (number of queries) of the output of* $\texttt{Reformulate}(q, \texttt{db})$ *is in* $O((6 * \#\texttt{db}^2)^{\#q})$*, with* #db *and* #$q$ *the sizes (number of triples) of* db *and q respectively.*

Proof of Theorem 5. As in the proof for Theorem 2, a close examination of the reformulation rules also exhibits producer-consumer dependencies among rules.

Given the possible forms of query triples that trigger reformulation rules, we write the reformulation schemes based on these dependencies using regular expressions as follows.

Query triple	Reformulation scheme
s y o .	$[(5).((9)+(10)+(11)+(12)+(13)).(14)^*.((15)+(16)).(17)^*]+[((6)+(7)+(8)).(17)^*]$
s rdf:type z .	$((9)+(10)+(11)+(12)+(13)).(14)^*.((15)+16)).(17)^*$
s rdf:type **c** .	$(14)^*.((15)+(16)).(17)^*$
s **p** o .	$(17)^*$

This said, we provide now an upper bound for the size of $\texttt{Reformulate}(q, \texttt{db})$, when q contains a single triple s y o ., s rdf:type z ., s rdf:type **c** ., or s **p** o . Assume $\texttt{db} = \langle \texttt{S}, \texttt{D} \rangle$ and let #S and #D the sizes (number of triples) of S and D respectively.

The triple s y o . can be either s y *val* . or s y z ., depending whether o is a value or a variable. In the former case, its reformulation scheme is reduced to $[(5).(14)^*.((15)+(16)).(17)^*]+[((6)+(7)+(8)).(17)^*]$, while in the latter case its reformulation scheme is that shown in the above table.

As for s y *val* .,

- reformulating s y *val* . with (5) leads to 1 triple of the form s rdf:type *val* ., which can be reformulated at most $2 * \#\texttt{S}$ times by the sequence $(14)^*.((15)+(16)).(17)^*$, i.e., at most #S times for $(14)^*.(15).(17)^*$ and for $(14)^*.(16).(17)^*$, as rules (14)–(17) are based on schema-level triples (there is at most #S schema-level triples in db). Summing up, the number of reformulations obtained starting from rule (5) is at most: $1 + 2 * \#\texttt{S}$.

- reformulating s y *val* . with rule (6) leads to at most #D triples of the form s **p** *val* . (there is at most #D instance-level triples in db). In turn, those triples can be reformulated at most #S times by the sequence $(17)^*$, as rule (17) is based on schema-level triples (there is at most #S schema-level triples in db). Summing up, the number of reformulations obtained starting from rule (6) is at most: $\#\texttt{D} * (1 + \#\texttt{S})$.

- reformulating s y *val* . with rules (7) and (8) leads to at most $2 * \#\texttt{S}$ triples of the form s **p** *val* . (there is at most #S schema-level triples in db, with at two properties per triple). In turn, those triples can be reformulated at most #S times by the sequence $(17)^*$, as rule (17) is based on schema-level triples (there is at most #S schema-level triples in db). Summing up, the number of reformulations obtained starting from rule (7) or (8) is at most: $2 * \#\texttt{S} * (1 + \#\texttt{S})$.

Summing up, the overall number of reformulations of `s` y val . is at most: $2 * \#\mathtt{S}^2 + 5 * \#\mathtt{S} + \#\mathtt{D} + 1$.

As for `s` y z .,

- reformulating `s` y z . with (5) leads to 1 triple of the form `s` rdf:type z . In turn, that triple can be reformulated using rule (9) in at most $\#\mathtt{D}$ triples of the form `s` rdf:type c . (there is at most $\#\mathtt{D}$ instance-level triples in `db`). The triple `s` rdf:type z . can also be reformulated using the rules (10)–(13) in at most $2 * \#\mathtt{S}$ triples of the form `s` rdf:type **c** . (there is at most $\#\mathtt{S}$ schema-level triples in `db`, with at most two classes per triple). Finally, the triples resulting from rules (9)–(13) can be reformulated at most $2 * \#\mathtt{S}$ times by the sequence $(14)^*.((15) + (16)).(17)^*$, i.e., at most $\#\mathtt{S}$ times for $(14)^*.(15).(17)^*$ and for $(14)^*.(16).(17)^*$, as rules (14)–(17) are based on schema-level triples (there is at most $\#\mathtt{S}$ schema-level triples in `db`). Summing up, the number of reformulations obtained starting from rule (5) is at most: $1 + (\#\mathtt{D} + 2 * \#\mathtt{S}) * (1 + 2 * \#\mathtt{S})$.

- reformulating `s` y z . with rule (6) leads to at most $\#\mathtt{D}$ triples of the form `s` **p** z . (there is at most $\#\mathtt{D}$ instance-level triples in `db`). In turn, those triples can be reformulated at most $\#\mathtt{S}$ times by the sequence $(17)^*$, as rule (17) is based on schema-level triples (there is at most $\#\mathtt{S}$ schema-level triples in `db`). Summing up, the number of reformulations obtained starting from rule (6) is at most: $\#\mathtt{D} * (1 + \#\mathtt{S})$.

- reformulating `s` y z . with rule (7) or (8) leads to at most $2 * \#\mathtt{S}$ triples of the form `s` **p** z . (there is at most $\#\mathtt{S}$ schema-level triples in `db`, with at two properties per triples). In turn, those triples can be reformulated at most $\#\mathtt{S}$ times by the sequence $(17)^*$, as rule (17) is based on schema-level triples (there is at most $\#\mathtt{S}$ schema-level triples in `db`). Summing up, the number of reformulations obtained starting from rule (7) or (8) is at most: $2 * \#\mathtt{S} * (1 + \#\mathtt{S})$.

Summing up, the overall number of reformulations of `s` y z . is at most: $6 * \#\mathtt{S}^2 + 3 * \#\mathtt{D} * \#\mathtt{S} + 2 * \#\mathtt{D} + 4 * \#\mathtt{S} + 1$.

We therefore get that the worst-case number of reformulations for `s` y `o` . is actually that of `s` y z .

By proceeding analogously with the other query triples, we show that the worst-case number of reformulations for a query triple is precisely that of `s` y z .

That is, for a query q made of a single triple, the overall upper bound for the size of the output of $\mathtt{Reformulate}(q, \mathtt{db})$ is: $1 + (6 * \#\mathtt{S}^2 + 3 * \#\mathtt{D} * \#\mathtt{S} + 2 * \#\mathtt{D} + 4 * \#\mathtt{S} + 1)$, where the leading 1 accounts for q itself.

The above result easily generalizes to a query of any size $\#q$. Given that $\#\mathtt{db} = \#\mathtt{S} + \#\mathtt{D}$, the size of the output of $\mathtt{Reformulate}(q, \mathtt{db})$ is in $O((6 * \#\mathtt{db}^2)^{\#q})$. □

Example 9 (Continued)**.** *The reformulation of the query* $q(x, y)$:- x rdf:type y . *w.r.t.* `db`*, asking for all resources and the classes to which they belong, is in Figure 14.*

$\texttt{Reformulate}^0(q, \texttt{db}) = \{q(x, y)\text{:-}\ x\ \text{rdf:type}\ y\ .\}$
$\texttt{Reformulate}^1(q, \texttt{db}) = \texttt{Reformulate}^0(q, \texttt{db}) \cup$
$\{q(x, \text{confP})\text{:-}\ x\ \text{rdf:type confP}\ .\ \ q(x, \text{posterCP})\text{:-}\ x\ \text{rdf:type posterCP}\ .$
$\ q(x, \_\text{:}b_0)\text{:-}\ x\ \text{rdf:type}\ \_\text{:}b_0\ .\ \ q(x, \text{paper})\text{:-}\ x\ \text{rdf:type paper}\ .$
$\ q(x, \text{conference})\text{:-}\ x\ \text{rdf:type conference}\ .\}$
$\texttt{Reformulate}^2(q, \texttt{db}) = \texttt{Reformulate}^1(q, \texttt{db}) \cup$
$\{q(x, \text{confP})\text{:-}\ x\ \text{rdf:type posterCP}\ .\ \ q(x, \text{confP})\text{:-}\ x\ \text{rdf:type}\ \_\text{:}b_0\ .$
$\ q(x, \text{confP})\text{:-}\ x\ \text{inProceedingsOf}\ z\ .\ \ q(x, \text{paper})\text{:-}\ x\ \text{rdf:type confP}\ .$
$\ q(x, \text{paper})\text{:-}\ x\ \text{hasTitle}\ z\ .\ \ q(x, \text{paper})\text{:-}\ x\ \text{hasAuthor}\ z\ .$
$\ q(x, \text{conference})\text{:-}\ z\ \text{inProceedingsOf}\ x\ .$
$\ q(x, \text{conference})\text{:-}\ x\ \text{hasName}\ z\ .\}$
$\texttt{Reformulate}^3(q, \texttt{db}) = \texttt{Reformulate}^2(q, \texttt{db}) \cup$
$\{q(x, \text{paper})\text{:-}\ x\ \text{rdf:type posterCP}\ .\ \ q(x, \text{paper})\text{:-}\ x\ \text{rdf:type}\ \_\text{:}b_0\ .$
$\ q(x, \text{paper})\text{:-}\ x\ \text{inProceedingsOf}\ z\ .\ \ q(x, \text{paper})\text{:-}\ x\ \text{hasContactA}\ z\ .\}$
$\texttt{Reformulate}^4(q, \texttt{db}) = \texttt{Reformulate}^3(q, \texttt{db})$

Figure 14: Sample reformulation of a query q w.r.t. the database of our running example.

7.2 From query reformulation to reformulation-based query answering

A requirement of any reformulation-based query answering technique is that the reformulated queries are equivalent to (or contained in) the original query (w.r.t. the database constraints), otherwise evaluating them might produce erroneous answers. It turns out that our query reformulation technique does not meet this requirement under the previously introduced definitions of evaluation and of answer set of a query against a database, as exemplified below.

Example 10 (Continued)**.** *Consider again the database* db*. For the previous query* $q(x, y)$*:-* x rdf:type y *., the evaluation of the queries in* $\texttt{Reformulate}(q, \texttt{db})$*, shown in Figure 14, yields erroneous answers. E.g., the tuple* $\langle \text{www2012}, \text{confP} \rangle$ *is an erroneous answer (see Example 7 for the correct answers) resulting from the evaluation against* db *of* $q(x, \text{confP})$*:-* x rdf:type $\_\text{:}b_0$. *in* $\texttt{Reformulate}^2(q, \texttt{db})$*, with the assignment* $\mu = \{x \rightarrow \text{www2012}, \_\text{:}b_0 \rightarrow \text{conference}\}$*.*

As the above example suggests, the issue is due to blank nodes. The semantics of blank nodes in BGP queries does not match the purpose for which they are brought into query reformulation by the Reformulate algorithm. Remember that the semantics of a blank node in a BGP query against a database is that of a non-distinguished variable. However, when our Reformulate algorithm brings a blank node in a query through a variable binding or a triple replacement, it refers precisely to that particular blank node in the database. In the above example, when $q(x, \text{confP})$:- x rdf:type $\_\text{:}b_0$. in $\texttt{Reformulate}^2(q, \texttt{db})$ is produced by rule (14) from $\_\text{:}b_0$ rdfs:subClassOf confP . $\in$ db and $q(x, \text{confP})$:- x rdf:type confP . in $\texttt{Reformulate}^2(q, \texttt{db})$, the goal is indeed to find conference paper values for x from the subclass $\_\text{:}b_0$ of confP.

Non-standard evaluation and answer set of a query against a database. To overcome the above issue, we introduce *alternate notions of evaluation and of answer*

set of a partially instantiated query against a database. The basic difference between the standard definitions from Section 7.1 and the alternate ones concerns blank nodes. Standard evaluation is based on assignments of $\texttt{VarBl}(q)$, all the query's variables and blank nodes, into database values. In contrast, the alternate definition only seeks to assign the query variables; blank nodes are left untouched, like URIs and literals.

Given a database $\texttt{db}$ whose set of values (URIs, blank nodes, literals) is $\texttt{Val}(\texttt{db})$ and a query $q_\sigma(\bar{x}_\sigma)$:- $(t_1, \ldots, t_\alpha)_\sigma$ whose set of variables (no blank nodes) is $\texttt{Var}(q_\sigma)$, the *non-standard evaluation* of q_σ against $\texttt{db}$ is defined as: $\tilde{q_\sigma}(\texttt{db}) = \{(\bar{x}_\sigma)_\mu \mid \mu : \texttt{Var}(q_\sigma) \rightarrow \texttt{Val}(\texttt{db})$ is a total assignment s.t. $((t_1, \ldots, t_\alpha)_\sigma)_\mu \subseteq \texttt{db}\}$.

The *non-standard answer set* of q_σ against $\texttt{db}$ is obtained by the non-standard evaluation of q_σ against $\texttt{db}^\infty$, which using our notation is denoted $\tilde{q_\sigma}(\texttt{db}^\infty)$.

The next Property shows how standard and non-standard definitions of query evaluation and answer set of queries are related. It follows directly from the fact that the assignments μ involved in non-standard evaluations, defined on $\texttt{Var}(q)$ only, are a subset of those allowed in standard evaluations, defined on $\texttt{VarBl}(q)$, as non-standard evaluations implicitly assign any URI, blank node, or literal to itself.

Property 2. *Let* $\texttt{db}$ *be a database and* q_σ *a (partially instantiated) query against* $\texttt{db}$*.*

1. $\tilde{q}_\sigma(\texttt{db}) \subseteq q_\sigma(\texttt{db})$ *and* $\tilde{q}_\sigma(\texttt{db}^\infty) \subseteq q_\sigma(\texttt{db}^\infty)$ *hold.*
2. *If* q_σ *does not contain blank nodes then* $\tilde{q}_\sigma(\texttt{db}) = q_\sigma(\texttt{db})$ *and* $\tilde{q}_\sigma(\texttt{db}^\infty) = q_\sigma(\texttt{db}^\infty)$*.*

With the above notion of non-standard evaluation in place, our reformulation-based query answering technique is:

Theorem 6. *Given a BGP query* q *without blank nodes and a database* $\texttt{db}$*, the following holds:*

$$q(\texttt{db}^\infty) = \bigcup_{q'_{\sigma'} \in \texttt{Reformulate}(q, \texttt{db})} \tilde{q}'_{\sigma'}(\texttt{db}).$$

Proof of Theorem 6. Let us first show that $q(\texttt{db}^\infty) \supseteq \bigcup_{q'_{\sigma'} \in \texttt{Reformulate}(q,\texttt{db})} \tilde{q}'_{\sigma'}(\texttt{db})$ holds. We actually show that $\tilde{q}'_{\sigma'}(\texttt{db}^\infty) \subseteq q(\texttt{db}^\infty)$ for any $q'_{\sigma'} \in \texttt{Reformulate}(q, \texttt{db})$, since $\tilde{q}'_{\sigma'}(\texttt{db}) \subseteq \tilde{q}'_{\sigma'}(\texttt{db}^\infty)$ (1. in Property 2). The proof is by induction on the length l of a sequence of reformulation rules leading to $q'_{\sigma'}$, starting from $\langle \texttt{db}, q \rangle$.

Base step. For $l = 0$, we have $q'_{\sigma'} = q$ and $\tilde{q}'_{\sigma'}(\texttt{db}^\infty) = q(\texttt{db}^\infty)$, since q is blank node free (2. in Property 2).

Inductive step. For $l < \alpha$, suppose that $\tilde{q}'_{\sigma'}(\texttt{db}^\infty) \subseteq q(\texttt{db}^\infty)$ holds. Now at $l = \alpha$, $q'_{\sigma'}$ has been produced from $q''_{\sigma''}$ by the application a given rule. In turn, $q''_{\sigma''}$ has been produced from q by a sequence of rules starting from $\langle \texttt{db}, q \rangle$. That sequence being of length $< \alpha$, we get $\tilde{q}''_{\sigma''}(\texttt{db}^\infty) \subseteq q(\texttt{db}^\infty)$ by the induction hypothesis. We show that $\tilde{q}'_{\sigma'}(\texttt{db}^\infty) \subseteq \tilde{q}''_{\sigma''}(\texttt{db}^\infty)$ to prove our claim. Let $\bar{x}_{\sigma'}$ be the (partially instantiated) output of $q'_{\sigma'}$ and $\bar{x}_{\sigma''}$ be that of $q''_{\sigma''}$.

- Consider the case where $q'_{\sigma'}$ is obtained from $q''_{\sigma''}$ by a rule of the form $\frac{\langle t_1 \in \texttt{db}, t_2 \in q_\sigma \rangle}{q_{\sigma \cup \nu}}$ that binds a variable of $q''_{\sigma''}$ using an assignment ν. Observe

here that $\sigma' = \sigma'' \cup \{\nu\}$ holds. If $(\bar{x}_{\sigma'})_\mu \in \tilde{q}'(\mathtt{db}^\infty)$, then $\mu \cup \{\nu\}$ is a total assignment of the variables of $q''_{\sigma''}$ such that $(\bar{x}_{\sigma'})_\mu = (\bar{x}_{\sigma''})_{\mu\cup\{\nu\}} \in \tilde{q}''_{\sigma''}(\mathtt{db}^\infty)$.

- Consider the case where $q'_{\sigma'}$ is obtained from $q''_{\sigma''}$ by a rule of the form $\frac{\langle t_1 \in \mathtt{db}, t_2 \in q_\sigma\rangle}{q_{\sigma[t_2/t_3]}}$ that replaces a triple in $q''_{\sigma''}$ by another one. Observe here that $\sigma' = \sigma''$ holds. If $(\bar{x}_{\sigma'})_\mu \in \tilde{q}'_{\sigma'}(\mathtt{db}^\infty)$, then $(t_{3\sigma'})_\mu \in \mathtt{db}^\infty$ and the immediate entailment rule $t_1, (t_{3\sigma'})_\mu \vdash^i_{\text{RDF}} (t_{2\sigma''})_\mu$ applies. As a result, $(t_{2\sigma''})_\mu \in \mathtt{db}^\infty$ and μ is a total assignment of the variables of $q''_{\sigma''}$ (that may also assign an extra variable generated by the rule leading from $q''_{\sigma''}$ to $q'_{\sigma'}$) such that $(\bar{x}_{\sigma'})_\mu = (\bar{x}_{\sigma''})_\mu \in \tilde{q}''_{\sigma''}(\mathtt{db}^\infty)$.

As there is no other form of rule that leads from $q''_{\sigma''}$ to $q'_{\sigma'}$, we get $\tilde{q}'_{\sigma'}(\mathtt{db}^\infty) \subseteq \tilde{q}''_{\sigma''}(\mathtt{db}^\infty)$ which concludes the proof of $q(\mathtt{db}^\infty) \supseteq \bigcup_{q'_{\sigma'} \in \mathtt{Reformulate}(q,\mathtt{db})} \tilde{q}'_{\sigma'}(\mathtt{db})$.

Let us show now that $q(\mathtt{db}^\infty) \subseteq \bigcup_{q'_{\sigma'} \in \mathtt{Reformulate}(q,\mathtt{db})} \tilde{q}'_{\sigma'}(\mathtt{db})$ holds. We actually show that $\tilde{q}_\sigma(\mathtt{db}^\infty) \subseteq \bigcup_{q'_{\sigma'} \in \mathtt{Reformulate}(q,\mathtt{db})} \tilde{q}'_{\sigma'}(\mathtt{db})$ holds with q_σ a possibly partially instantiated query, for which $q(\mathtt{db}^\infty) \subseteq \bigcup_{q'_{\sigma'} \in \mathtt{Reformulate}(q,\mathtt{db})} \tilde{q}'_{\sigma'}(\mathtt{db})$ is a special case when q_σ is not partially instantiated ($\sigma = \emptyset$) and does no contain blank nodes ($\tilde{q}(\mathtt{db}^\infty) = q(\mathtt{db}^\infty)$, 2. in Property 2).

Provided that q_σ is of the form $q(\bar{x}_\sigma)$:- $(t_1, \ldots, t_n)_\sigma$, suppose that $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}_\sigma(\mathtt{db}^\infty)$ and let us show that there exists $q'_{\sigma'} \in \mathtt{Reformulate}(q, \mathtt{db})$ such that $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}'_{\sigma'}(\mathtt{db})$. We show this by induction on the length l of a (minimal) sequence of immediate entailment rules such that $((t_1, \ldots, t_n)_\sigma)_\mu \subseteq \mathtt{db}^l$: for any l there exists a (possibly empty) sequence of reformulation rules leading to $q'_{\sigma'}$, starting from $\langle \mathtt{db}, q_\sigma \rangle$.

Base step. For $l = 0$, we have $((t_1, \ldots, t_n)_\sigma)_\mu \subseteq \mathtt{db}^0$, thus $((t_1, \ldots, t_n)_\sigma)_\mu \subseteq \mathtt{db}$, so for the empty sequence of reformulation rules we have $q'_{\sigma'} = q_\sigma$, and $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}'_{\sigma'}(\mathtt{db})$.

Inductive step. For $l < \alpha$, suppose that the above claim holds. Now at $l = \alpha$, for $1 \leq i \leq n$, $t_{i\sigma}$ matches $(t_{i\sigma})_\mu$: it is either $(t_{i\sigma})_\mu$ or a generalization of $(t_{i\sigma})_\mu$ using one, two, or three (distinct) variables. Moreover, $(t_{i\sigma})_\mu$ has been added to the saturation at $l \leq \alpha$ by an entailment rule of Figure 9. Indeed, the entailment rules of the other Figures do not need to be considered due to Theorem 1.

Consider the case of the rule: $\mathtt{s}$ rdfs:subClassOf $\mathtt{o}$., $\mathtt{s}_1$ rdf:type $\mathtt{s}$. $\vdash^i_{\text{RDF}}$ $\mathtt{s}_1$ rdf:type $\mathtt{o}$. Assume that $(t_{i\sigma})_\mu = v$ rdf:type c_1 ., i.e., has been produced from $\{c_2$ rdfs:subClassOf c_1 ., v rdf:type c_2 .$\} \subseteq \mathtt{db}^{\alpha-1}$. Observe that v, c_1, and c_2 can be blank nodes.

- If $t_{i\sigma} = v$ rdf:type c_1 . then consider the query $q''_{\sigma''}$ obtained from q using the reformulation rule (14), in which v rdf:type c_1 . is replaced by v rdf:type c_2 . As a result, $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}''(\mathtt{db}^{\alpha-1})$ and, by the induction hypothesis, there exists a sequence of reformulation rules leading to $q'_{\sigma'}$ starting from $\langle \mathtt{db}, q''_{\sigma''} \rangle$, thus from $\langle \mathtt{db}, q_\sigma \rangle$, such that $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}'_{\sigma'}(\mathtt{db})$.
- If $t_{i\sigma} = x$ rdf:type c_1 . then consider the query $q''_{\sigma''}$ obtained from q using the reformulation rule (14), in which x rdf:type c_1 . is replaced by x rdf:type c_2 . As a result, $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}''(\mathtt{db}^{\alpha-1})$ and, by the induction hypothesis, there exists a sequence of reformulation rules leading to $q'_{\sigma'}$ starting from $\langle \mathtt{db}, q''_{\sigma''} \rangle$, thus from $\langle \mathtt{db}, q_\sigma \rangle$, such that $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}'_{\sigma'}(\mathtt{db})$.

- If $t_{i\sigma} = v$ rdf:type x . then consider the query $q''_{\sigma''}$ obtained from q using the reformulation rules (11) then (14), in which v rdf:type x . is replaced by x rdf:type c_2 . As a result, $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}''(\mathtt{db}^{\alpha-1})$ and, by the induction hypothesis, there exists a sequence of reformulation rules leading to $q'_{\sigma'}$ starting from $\langle \mathtt{db}, q''_{\sigma''} \rangle$, thus from $\langle \mathtt{db}, q_\sigma \rangle$, such that $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}'_{\sigma'}(\mathtt{db})$.

- If $t_{i\sigma} = v\ x\ c_1$. then consider the query $q''_{\sigma''}$ obtained from q using the reformulation rules (5) then (14), in which $v\ x\ c_1$. is replaced by v rdf:type c_2 . As a result, $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}''(\mathtt{db}^{\alpha-1})$ and, by the induction hypothesis, there exists a sequence of reformulation rules leading to $q'_{\sigma'}$ starting from $\langle \mathtt{db}, q''_{\sigma''} \rangle$, thus from $\langle \mathtt{db}, q_\sigma \rangle$, such that $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}'_{\sigma'}(\mathtt{db})$.

- If $t_{i\sigma} = x\ y\ c_1$. then consider the query $q''_{\sigma''}$ obtained from q using the reformulation rules (5) then (14), in which $x\ y\ c_1$. is replaced by x rdf:type c_2 . As a result, $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}''(\mathtt{db}^{\alpha-1})$ and, by the induction hypothesis, there exists a sequence of reformulation rules leading to $q'_{\sigma'}$ starting from $\langle \mathtt{db}, q''_{\sigma''} \rangle$, thus from $\langle \mathtt{db}, q_\sigma \rangle$, such that $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}'_{\sigma'}(\mathtt{db})$.

- If $t_{i\sigma} = x$ rdf:type y . then consider the query $q''_{\sigma''}$ obtained from q using the reformulation rules (11) then (14), in which x rdf:type y . is replaced by x rdf:type c_2 . As a result, $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}''(\mathtt{db}^{\alpha-1})$ and, by the induction hypothesis, there exists a sequence of reformulation rules leading to $q'_{\sigma'}$ starting from $\langle \mathtt{db}, q''_{\sigma''} \rangle$, thus from $\langle \mathtt{db}, q_\sigma \rangle$, such that $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}'_{\sigma'}(\mathtt{db})$.

- If $t_{i\sigma} = v\ x\ y$. then consider the query $q''_{\sigma''}$ obtained from q using the reformulation rules (5), then (11) then (14), in which $v\ x\ y$. is replaced by v rdf:type c_2 . As a result, $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}''(\mathtt{db}^{\alpha-1})$ and, by the induction hypothesis, there exists a sequence of reformulation rules leading to $q'_{\sigma'}$ starting from $\langle \mathtt{db}, q''_{\sigma''} \rangle$, thus from $\langle \mathtt{db}, q_\sigma \rangle$, such that $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}'_{\sigma'}(\mathtt{db})$.

- If $t_{i\sigma} = x\ y\ z$. then consider the query $q''_{\sigma''}$ obtained from q using the reformulation rules (5), then (11) then (14), in which $x\ y\ z$. is replaced by x rdf:type c_2 . As a result, $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}''(\mathtt{db}^{\alpha-1})$ and, by the induction hypothesis, there exists a sequence of reformulation rules leading to $q'_{\sigma'}$ starting from $\langle \mathtt{db}, q''_{\sigma''} \rangle$, thus from $\langle \mathtt{db}, q_\sigma \rangle$, such that $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}'_{\sigma'}(\mathtt{db})$.

The rest of the proof is omitted here as it amounts to show, similarly as above, that the claim also holds at $l = \alpha$ for the three other entailment rules of Figure 9. □

Note that Theorem 6 considers queries without blank nodes. This assumption is made *without loss of generality*, since blank nodes act as (thus can be equivalently replaced by) non-distinguished variables in BGP queries. We make this simplifying assumption so that at the end of the reformulation step, we do not have two kinds (thus two different treatments in query evaluation) of blank nodes: those coming from the original BGP query (i.e., non-distinguished variables) and those coming from the reformulation step (i.e., values referring to themselves in the database).

At a practical level, Theorem 6 reads as follows: to answer a query q against a database db, it suffices to (*i*) reformulate q w.r.t. db and (*ii*) evaluate each reformulated query on the *original* database db, using the *non-standard* evaluation. In other words, query reformulation (based on db) and non-standard

evaluation of partially instantiated queries allow computing the exact answer set, *without* actually saturating the database. Importantly, based on Section 3, reformulation-based query answering can be delegated to *any* RDBMS by storing db in a Triple table, and then by evaluating queries using relational evaluation, *without replacing blank nodes by fresh non-distinguished variables.*

Example 11 (Continued)**.** *Consider again the query* $q(x, y)$*:-* x rdf:type y . *The answer set of* q *against the previous database* db *is:*

$$\bigcup_{q_\sigma \in \texttt{Reformulate}(q,\texttt{db})} \tilde{q}_\sigma(\texttt{db}) = \{\langle \text{doi}_1, \_{:}b_0\rangle, \langle \text{www2012}, \text{conference}\rangle, \langle \text{doi}_1, \text{confP}\rangle, \langle \text{doi}_1, \text{paper}\rangle, \langle \_{:}b_2, \text{conference}\rangle\}.$$

Note that this answer set coincides with that obtained by saturation-based query answering in Example 7.

8 Experimental evaluation

We implemented our Saturate and Reformulate algorithms in Java 1.6 and deployed them on top of PostgreSQL v8.5 (using standard default settings).

8.1 Settings

Instance-level triples are stored in a Triple(s, p, o) table (Section 3). The set-based saturation is stored in a Sat(s, p, o) table, while the multiset-based saturation (allowing maintenance) is stored in a SatM(s, p, o, isDerived, count) table (Section 6). Each table is indexed by all permutations of the (s, p, o) columns, leading to a total of 6 indexes; the spo index is clustering. We adopted this indexing choice (inspired by [16]) to give PostgreSQL efficient query evaluation opportunities. *Schema-level* triples are kept in memory. *All measured times are averaged over* 10 *executions.*

We denote $\mathbf{t_{sat}}$ the time to saturate a given database by Algorithm Saturate (Section 6.1), and $\mathbf{t_{sat+}}$ the time to saturate it by Algorithm $\texttt{Saturate}_+$ (Section 6.2).

8.2 Saturation

We evaluated our techniques on the graphs presented in Table 1. Saturation added between 10% and 41% to the database size. The column #Multiset counts: the explicit triples, plus each entailed triple, *counted as many times as it is derived (entailed).* As "Multiset increase" shows, this more than doubles the size of each graph as entailed triples can be derived in several ways. Table 1 also shows $\mathbf{t_{sat}}$ and $\mathbf{t_{sat+}}$ for each graph. As expected, Saturate is (slightly) faster than $\texttt{Saturate}_+$. However, if the graph is updated, one can maintain the saturation only if $\texttt{Saturate}_+$ was used, as explained in Section 6.2.

The size of the graphs makes it difficult for a Java program to process the whole data in memory in one pass. Therefore, we propose a partition-based

Graph	Barton [24]	DBpedia [25]	DBLP [9]
#Schema	101	5,666	41
#Instance	33,912,142	26,988,098	8,424,214
#Saturation	38,969,023	29,861,597	11,882,398
Saturation increase (%)	14.91	10.65	41.05
#Multiset	73,551,358	66,029,147	18,684,182
Multiset increase (%)	116.89	227.37	121.79
t_{sat} (s)	4,294	2,742	900
t_{sat+} (s)	4,586	2,977	996

Table 1: Graph characteristics; time to saturate.

saturation method, which reads the graphs in partitions of a specified number of triples at a time. The positive logic nature of RDFS makes implementing incremental updates close to trivial [12]. Our partition-based approach can be seen as consecutive incremental updates made on the saturation table.

The time to saturate each graph using different partition sizes is shown in Figure 15. In the following experiments we will consider the saturation time obtained when reading the data in partitions of 500,000 triples, values shown in Table 1.

8.3 Query answering times

We hand-picked a set of 26 queries for the DBLP graph aiming at a variety of behavior when reformulated against the DBLP schema. The queries have between 1 and 10 triple patterns (6 on average); all have non-empty results. Similarly, we chose 17 queries for the Barton graph and 21 queries for the DBpedia graph. The queries are detailed in Tables 2-4.

We obtained similar query answering times on the two saturation tables, `Sat` and `SatM`, therefore from this point further only the `SatM` results will be discussed. For a query q, we denote by $\mathbf{t^{sat}}(q)$ the time to answer q against the *already saturated* database `SatM`, and $\mathbf{t^{ref}}(q)$ the time to answer q through reformulation, i.e., reformulating q and evaluating its reformulation against the `Triple` table.

Figure 16 shows, for each query: the number of union terms in the reformulated query (in parentheses after the query name); the time $\mathbf{t^{sat}}(q)$; the time $\mathbf{t^{ref}}(q)$; the sum $\mathbf{t^{sat}}(q) + \mathbf{t_{sat+}}$. As expected, $\mathbf{t^{sat}}(q)$ is significantly smaller than $\mathbf{t^{ref}}(q)$ for queries with large reformulations. However, if one factors in the saturation time $\mathbf{t_{sat+}}$, the saturation-based approach becomes expensive. Inspecting the results, we also found small-result queries have small $\mathbf{t^{sat}}$ and $\mathbf{t^{ref}}$, an encouraging sign that PostgreSQL's optimizer handled correctly both the original and the reformulated queries.

8.4 Update performance

We now compare our saturation and reformulation techniques upon graph updates. Updates have no impact on reformulation, but saturation needs to maintain `SatM`. To measure this overhead, we performed updates on the data, respectively, on the schema, by adding one triple, respectively, removing one triple from each. The target of our update (inserted, respectively, deleted triple) is

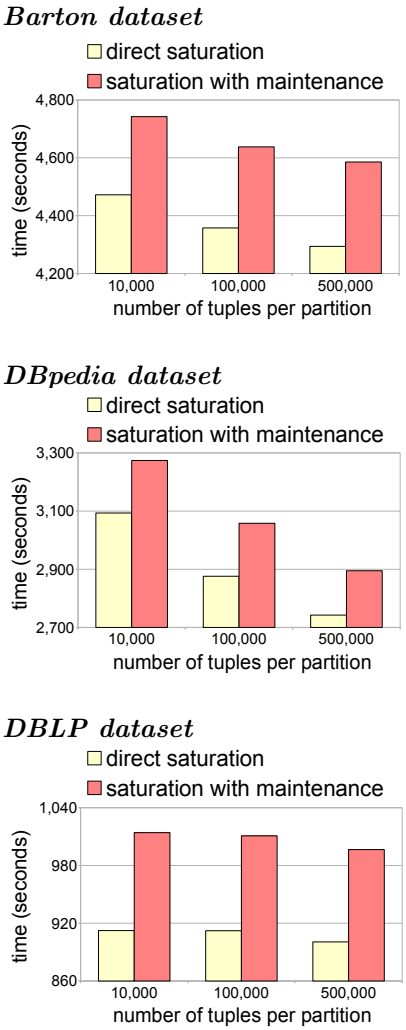


Figure 15: Saturation times.

chosen randomly based on the existing triples in the database: the triple we insert does not belong to the database, while for deletions, we ensure that we delete a triple which was present in the database. Moreover, the triples we added and removed in this experiment each have a distinct property value. As updates on the data, we ran overall 42 insertions and 42 deletions over the DBLP dataset, and 40 of each for the Barton and DBpedia datasets. For what concerns the schema updates, in the DBLP schema, we successively deleted each of the 41 schema triples (the schema was restored between every two deletions). For Barton we performed 26 schema deletions and 39 for DBpedia. The triples we added to the schemas were obtained by picking existing schema triples and slightly modifying them, changing their object value to ensure the added triple was not already in the schema.

Figures 17 and 18 show the time to insert into, respectively delete from, and maintain `SatM` for each of the considered schema triples. Under each time bar,

(143)	$Q_{01}(x, y)$	:- x rdf:type y .
(46)	$Q_{02}(x)$	:- x rdf:type *mods*:*Item* .
(12)	$Q_{03}(x)$	:- x rdf:type *mods*:*Name* .
(1)	$Q_{04}(x)$	:- x rdf:type *mods*:*Text* .
(2)	$Q_{05}(x, y)$	:- x *mods*:*language* y .
(2)	$Q_{06}(x, y)$	:- x *mods*:*description* y .
(2)	$Q_{07}(x, z)$	:- x rdf:type *mods*:*Text* ., x *mods*:*language* z .
(414)	$Q_{08}(y, z)$	:- x rdf:type *mods*:*Text* ., x y z ., x *mods*:*language* *language*:*fre* .
(2)	$Q_{09}(x, z)$	:- x rdf:type *mods*:*Text* ., x *role*:*creator* z ., x *mods*:*language* *language*:*fre* .
(143)	$Q_{10}(x, y)$	:- x *mods*:*origin* *info*:*DLC* ., x *mods*:*records* z ., z rdf:type y .
(46)	$Q_{11}(x, z)$	:- x *mods*:*origin* *info*:*DLC* ., x *mods*:*records* z ., z rdf:type *mods*:*Item* .
(1)	$Q_{12}(x, z)$	:- x *mods*:*origin* *info*:*DLC* ., x *mods*:*records* z ., z rdf:type *mods*:*Text* .
(176)	$Q_{13}(y)$	:- x y z ., x *mods*:*records* t ., t rdf:type *mods*:*Text* .
(1)	$Q_{14}(z)$	:- x *mods*:*created* z ., x *mods*:*records* t ., t rdf:type *mods*:*Text* .
(143)	$Q_{15}(x, y, z)$	:- x *mods*:*point* "*end*" ., x *mods*:*encoding* y ., x rdf:type z .
(46)	$Q_{16}(x, y)$	:- x *mods*:*point* "*end*" ., x *mods*:*encoding* y ., x rdf:type *mods*:*Item* .
(9)	$Q_{17}(x, y)$	:- x *mods*:*point* "*end*" ., x *mods*:*encoding* y ., x rdf:type *mods*:*Date* .

mods	http://simile.mit.edu/2006/01/ontologies/mods3#
language	http://simile.mit.edu/2006/01/language/iso639-2b/
role	http://simile.mit.edu/2006/01/role/
info	info:marcorg/

Table 2: Barton queries and the size of their reformulation.

we list in parenthesis after the added/deleted triple name the triple property: “sp" for rdfs:subPropertyOf, “sc" for rdfs:subClassOf, “domain" for rdfs:domain and “range" for rdfs:range. The last column in each graph is an average over the observed triple insertion times.

The figures demonstrate that despite our careful implementation, index usage etc., the maintenance of `SatM` when the schema changes is quite expensive; moreover, maintenance in the case of deletions is at least one order of magnitude slower than in the case of insertions. This illustrates the inherent difficulty raised by the recursive process required to analyze the consequences of withdrawing one schema statement. Interestingly, modifying the schema of the DBpedia dataset incurs significantly lower costs than in the case of Barton and DBLP. This is because the DBpedia schema is simpler, in particular it does not feature rdfs:subPropertyOf statements.

The graphs in Figure 19 summarize our findings on the maintenance required by data and schema updates. The graphs show:

(9793)	$Q_{01}(x, y)$	:- *resource*:*France* x y .
(8188)	$Q_{02}(x, y)$	:- x rdf:type y .
(8188)	$Q_{03}(x)$	:- *resource*:*France* rdf:type x .
(2220)	$Q_{04}(x)$	:- x rdf:type *owl*:*Thing* .
(463)	$Q_{05}(x)$	:- x rdf:type *ontology*:*Person* .
(347)	$Q_{06}(x)$	:- x rdf:type *ontology*:*Organisation* .
(39)	$Q_{07}(x)$	:- x rdf:type *ontology*:*Company* .
(11)	$Q_{08}(x)$	:- x rdf:type *ontology*:*Animal* .
(1)	$Q_{09}(x)$	:- *resource*:*France ontology*:*currency* x .
(9793)	$Q_{10}(x, y, z)$	:- *resource*:*France* x y ., y *foaf*:*name* z .
(2229)	$Q_{11}(x, y, z)$	:- y x *ontology*:*Place* ., y *foaf*:*name* z .
(463)	$Q_{12}(x)$	:- x rdf:type *ontology*:*Person* ., *resource*:*Cubix ontology*:*starring* x .
(347)	$Q_{13}(x, y)$	:- x *foaf*:*homepage* y ., x rdf:type *ontology*:*Organisation* .
(39)	$Q_{14}(y, x)$	:- y rdf:type *ontology*:*Company* ., y *ontology*:*headquarter* x .
(1)	$Q_{15}(y, x)$	:- y *foaf*:*name* x ., y *foaf*:*page resource*:*Trinity* .
(9793)	$Q_{16}(x, y, z, t)$	:- *resource*:*Eurosport* x y ., *resource*:*Eurosport ontology*:*country* z ., *resource*:*Eurosport* *foaf*:*homepage* t .
(463)	$Q_{17}(y, z)$	:- y rdf:type *ontology*:*Person* ., *resource*:*Cubix ontology*:*starring* y ., y *ontology*:*occupation* z .
(39)	$Q_{18}(x, y, z)$	:- x rdf:type *ontology*:*Company* ., x *ontology*:*headquarter* y ., x *foaf*:*homepage* z .
(1)	$Q_{19}(x, y, z)$	:- *resource*:*Eurosport* *foaf*:*name* x ., *resource*:*Eurosport ontology*:*country* y ., *resource*:*Eurosport* *foaf*:*homepage* z .
(463)	$Q_{20}(x, y, z)$	:- y rdf:type *ontology*:*Person* ., y *ontology*:*nationality* z ., y *ontology*:*occupation resource*:*Author* ., y *ontology*:*hometown* x .
(39)	$Q_{21}(x, y, z)$	:- x rdf:type *ontology*:*Company* ., x *ontology*:*headquarter* y ., x *ontology*:*alliance resource*:*Star_Alliance* ., x *foaf*:*homepage* z .

resource	http://dbpedia.org/resource/
ontology	http://dbpedia.org/ontology/
foaf	http://xmlns.com/foaf/0.1/
owl	http://www.w3.org/2002/07/owl#

Table 3: DBpedia queries and the size of their reformulation.

- the average time to insert into (resp. delete from) the `Triple` table (we do not show the detailed times since they were all very similar to each other and thus close to the average, plotted in Figure 19)
- the average time to insert into (resp. delete from) *and maintain* the `SatM` table (the same comment applies)
- the average time to *maintain* the `SatM` table after an insertion (resp. deletion) made on the schema.

Recall that `SatM` triples may be simply updated e.g. to increase/decrease a derivation count. We see that handling `SatM` is slower than updating `Triple`, by one order of magnitude for instance insertions, and two orders of magnitude for instance deletions. This shows that while the saturation algorithm $\texttt{Saturate}_+$

(121)	$Q_{01}(x,y,z)$	:- x y z .
(684)	$Q_{02}(x,y)$	:- x rdf:type *dblp:Document* ., x *dblp:datatypeField* y .
(36)	$Q_{03}(x)$	:- x rdf:type *dblp:Document* ., x *elm:publisher* "*Springer*" .
(1)	$Q_{04}(x)$	:- x rdf:type *dblp:Book* ., x *elm:publisher* "*Springer*" .
(1)	$Q_{05}(y,z)$	:- x rdf:type *foaf:Person* ., x *foaf:name* y ., x *foaf:homepage* z .
(19)	$Q_{06}(y,u,t)$	:- x *dblp:datatypeField* "*Algorithmica*" ., x *elm:title* y ., x *elm:creator* u ., x *elm:date* t .
(4)	$Q_{07}(y,u,t)$	:- x *dblp:objectField* "*Algorithmica*" ., x *elm:title* y ., x *elm:creator* u ., x *elm:date* t .
(19)	$Q_{08}(u,z)$	:- x *dblp:editor* y ., y *foaf:name* z ., x *elm:title* u ., x *dblp:datatypeField* v ., x *elm:publisher* "*Springer*" .
(4)	$Q_{09}(u,z)$	:- x *dblp:editor* y ., y *foaf:name* z ., x *elm:title* u ., x *dblp:objectField* v ., x *elm:publisher* "*Springer*" .
(138)	$Q_{10}(t,n,m)$	:- x rdf:type *dblp:Document* ., x *dblp:editor* y ., x *elm:creator* z ., y *foaf:name* n ., z *foaf:name* m ., x *elm:title* t ., x *dblp:objectField* u .
(36)	$Q_{11}(t,y,j,x)$	:- p *elm:creator* a ., a *foaf:name* "*Alison Cawsey*" ., p *elm:title* t ., p *dblp:year* y ., p *dblp:pages* x ., p *dblp:journal* j ., p rdf:type *dblp:Document* .
(36)	$Q_{12}(t,y,x)$	:- p *elm:creator* a ., a *foaf:name* "*Hugh Darwen*" ., p *elm:title* t ., p *dblp:year* y ., p *dblp:pages* x ., p *dblp:journal* "*SIGMOD Record*" ., p rdf:type *dblp:Document* .
(36)	$Q_{13}(t,j,x)$	:- p *elm:creator* a ., a *foaf:name* "*Dana Randall*" ., p *elm:title* t ., p *dblp:year* "2006" ., p *dblp:pages* x ., p *dblp:journal* j ., p rdf:type *dblp:Document* .
(36)	$Q_{14}(n,t,x)$	:- p *elm:creator* a ., a *foaf:name* n ., p *elm:title* t ., p *dblp:year* "1966" ., p *dblp:pages* x ., p *dblp:journal* "*Information and Control*" ., p rdf:type *dblp:Document* .
(1)	$Q_{15}(t,y,j,x)$	:- p *elm:creator* a ., a *foaf:name* "*Alison Cawsey*" ., p *elm:title* t ., p *dblp:year* y ., p *dblp:pages* x ., p *dblp:journal* j ., p rdf:type *dblp:Article* .
(1)	$Q_{16}(t,y,x)$	:- p *elm:creator* a ., a *foaf:name* "*Hugh Darwen*" ., p *elm:title* t ., p *dblp:year* y ., p *dblp:pages* x ., p *dblp:journal* "*SIGMOD Record*" ., p rdf:type *dblp:Article* .
(1)	$Q_{17}(t,j,x)$	:- p *elm:creator* a ., a *foaf:name* "*Dana Randall*" ., p *elm:title* t ., p *dblp:year* "2006" ., p *dblp:pages* x ., p *dblp:journal* j ., p rdf:type *dblp:Article* .
(1)	$Q_{18}(n,t,x)$	:- p *elm:creator* a ., a *foaf:name* n ., p *elm:title* t ., p *dblp:year* "1966" ., p *dblp:pages* x ., p *dblp:journal* "*Information and Control*" ., p rdf:type *dblp:Article* .
(36)	$Q_{19}(n,t,k,p,x)$	:- b *elm:creator* a ., a *foaf:name* n ., b *elm:title* t ., b *dblp:year* "1991" ., b *dblp:pages* x ., b *dblp:booktitle* k ., b *elm:publisher* p ., b rdf:type *dblp:Document* .
(36)	$Q_{20}(t,y,p,x)$	:- b *elm:creator* a ., a *foaf:name* "*William J. Frawley*" ., b *elm:title* t ., b *dblp:year* y ., b *dblp:pages* x ., b *dblp:booktitle* "*Knowledge Discovery in Databases*" ., b *elm:publisher* p ., b rdf:type *dblp:Document* .
(1)	$Q_{21}(n,t,k,p,x)$	:- b *elm:creator* a ., a *foaf:name* n ., b *elm:title* t ., b *dblp:year* "1991" ., b *dblp:pages* x ., b *dblp:booktitle* k ., b *elm:publisher* p ., b rdf:type *dblp:Book* .
(1)	$Q_{22}(t,y,p,x)$	:- b *elm:creator* a ., a *foaf:name* "*William J. Frawley*" ., b *elm:title* t ., b *dblp:year* y ., b *dblp:pages* x ., b *dblp:booktitle* "*Knowledge Discovery in Databases*" ., b *elm:publisher* p ., b rdf:type *dblp:Book* .
(36)	$Q_{23}(t,y,j,x)$	:- p *elm:creator* a_1 ., a_1 *foaf:name* "*Grzegorz Rozenberg*" ., p *elm:creator* a_2 ., a_2 *foaf:name* "*Azriel Rosenfeld*" ., p *elm:title* t ., p *dblp:year* y ., p *dblp:pages* x ., p *dblp:journal* j ., p rdf:type *dblp:Document* .
(1)	$Q_{24}(t,y,j,x)$	:- p *elm:creator* a_1 ., a_1 *foaf:name* "*Grzegorz Rozenberg*" ., p *elm:creator* a_2 ., a_2 *foaf:name* "*Azriel Rosenfeld*" ., p *elm:title* t ., p *dblp:year* y ., p *dblp:pages* x ., p *dblp:journal* j ., p rdf:type *dblp:Article* .
(36)	$Q_{25}(t,y,k,p,x)$	:- b *elm:creator* a_1 ., a_1 *foaf:name* "*Christopher J. Matheus*" ., b *elm:creator* a_2 ., a_2 *foaf:name* "*William J. Frawley*" ., b *elm:title* t ., b *dblp:year* y ., b *dblp:pages* x ., b *dblp:booktitle* k ., b *elm:publisher* p ., b rdf:type *dblp:Document* .
(1)	$Q_{26}(t,y,k,p,x)$	:- b *elm:creator* a_1 ., a_1 *foaf:name* "*Christopher J. Matheus*" ., b *elm:creator* a_2 ., a_2 *foaf:name* "*William J. Frawley*" ., b *elm:title* t ., b *dblp:year* y ., b *dblp:pages* x ., b *dblp:booktitle* k ., b *elm:publisher* p ., b rdf:type *dblp:Book* .

elm	http://purl.org/dc/elements/1.1/
foaf	http://xmlns.com/foaf/0.1/
dblp	http://sw.deri.org/ aharth/2004/07/dblp/dblp.owl#

Table 4: DBLP queries and the size of their reformulation.

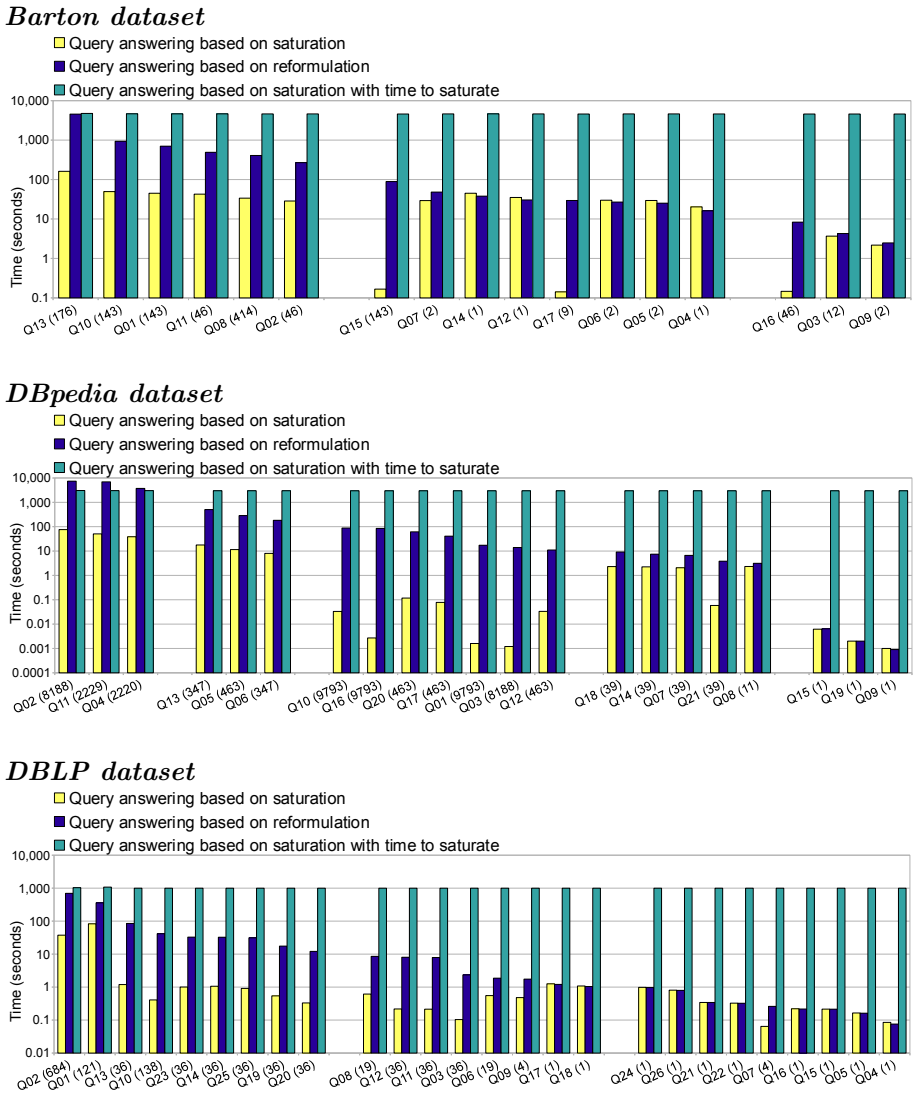


Figure 16: Query answering times.

and the `SatM` table are required in order to avoid saturating from scratch, saturation maintenance may get costly due to the recursive nature of entailment. In particular, in the case of schema updates, maintaining the saturation may sometimes be more costly than re-saturating.

8.5 Saturation thresholds

We now study when saturation pays off over multiple query runs. We call the *saturation threshold* of a query q, or $\mathbf{st}(q)$, the smallest integer n such that:

$$n \times \mathbf{t^{ref}}(q) > n \times \mathbf{t^{sat}}(q) + \mathbf{t_{sat+}}$$

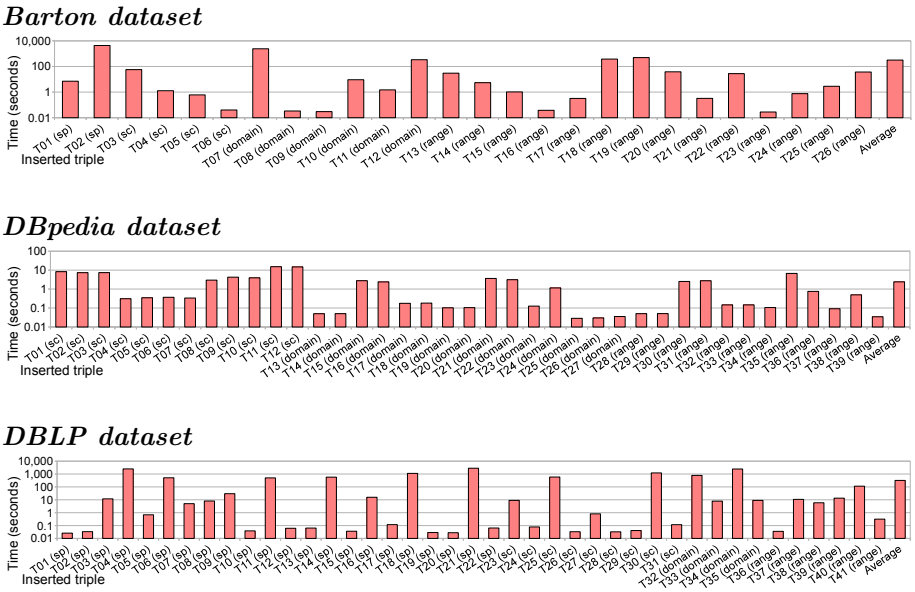


Figure 17: Schema triple insertion times.

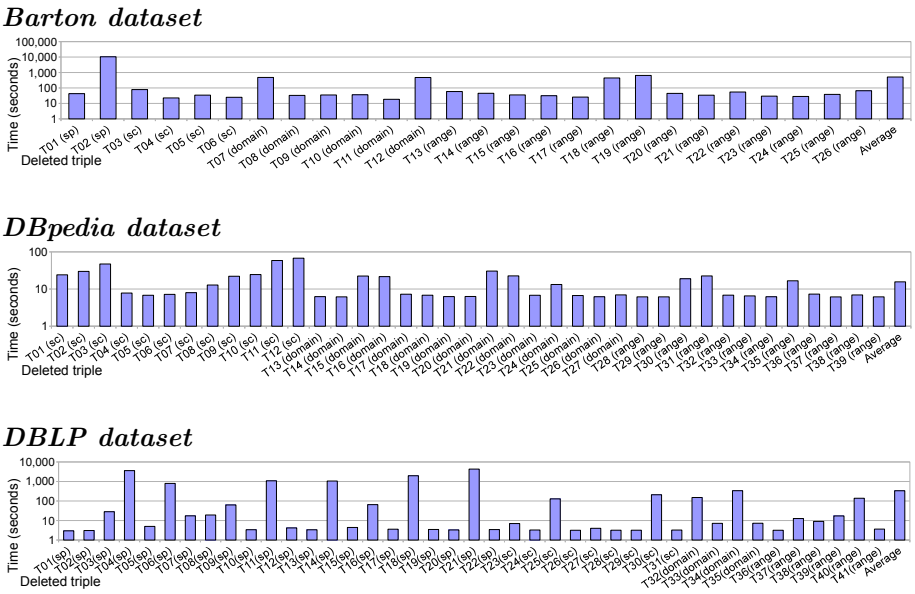


Figure 18: Schema triple deletion times.

In other words, n is the minimum number of times one needs to run q in order for the whole saturation cost to amortize.

Similarly, we study how many times q should run in order for the *maintenance overhead due to one instance or schema update* to pay off. We formalize this as follows. Let $\mathbf{t}^{+}_{\texttt{Triple}}$ be the time to insert one statement in `Triple`, and $\mathbf{t}^{+}_{\texttt{SatM}}$ be the time to propagate the insertion of one triple to the `SatM`

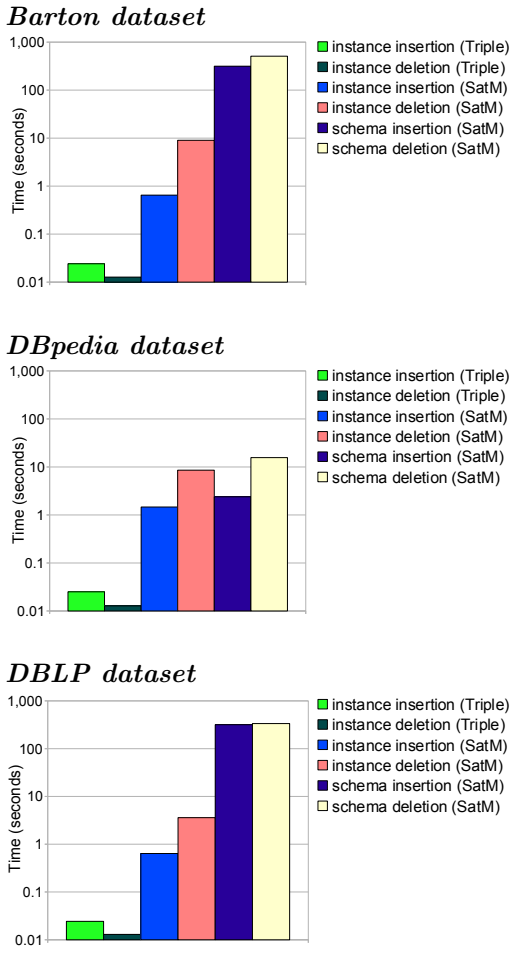


Figure 19: Update times.

relation. Then, the *saturation threshold for an instance insertion*, denoted $\mathbf{st}_{\mathbf{i}}^{+}(q)$, is the smallest n for which: $n \times \mathbf{t}^{\mathbf{ref}}(q) + \mathbf{t}^{+}_{\texttt{Triple}} > n \times \mathbf{t}^{\mathbf{sat}}(q) + \mathbf{t}^{+}_{\texttt{SatM}}$. In other words, $\mathbf{st}_{\mathbf{i}}^{+}(q)$ is the minimum number of times one needs to run q in order for the maintenance overhead due to the insertion of one triple (recall the graphs in Figure 19) to amortize. We similarly define the *saturation threshold for an instance deletion*, denoted $\mathbf{st}_{\mathbf{i}}^{-}(q)$, as the smallest n for which: $n \times t^{ref}(q) + t^{-}_{\texttt{Triple}} > n \times t^{sat}(q) + t^{-}_{\texttt{SatM}}$.

Schema updates do not affect the `Triple` table, since the schema is kept in memory, but they can have a big impact on `SatM`. Similar to $\mathbf{st}(q)$, we define the *saturation threshold for a schema insertion* $\mathbf{st}_{\mathbf{s}}^{+}(q)$ and *deletion* $\mathbf{st}_{\mathbf{s}}^{-}(q)$, as the minimum number of times one needs to run q in order for the schema update cost to amortize.

Figure 20 shows the 5 saturation thresholds for our queries. The vertical (log-scale) axis is limited to 10^5 (Barton) and 10^4 (DBpedia and DBLP) for readability. The thresholds follow a similar trend, strongly determined by the size of the reformulated query (shown in parentheses on the x axis). The larger

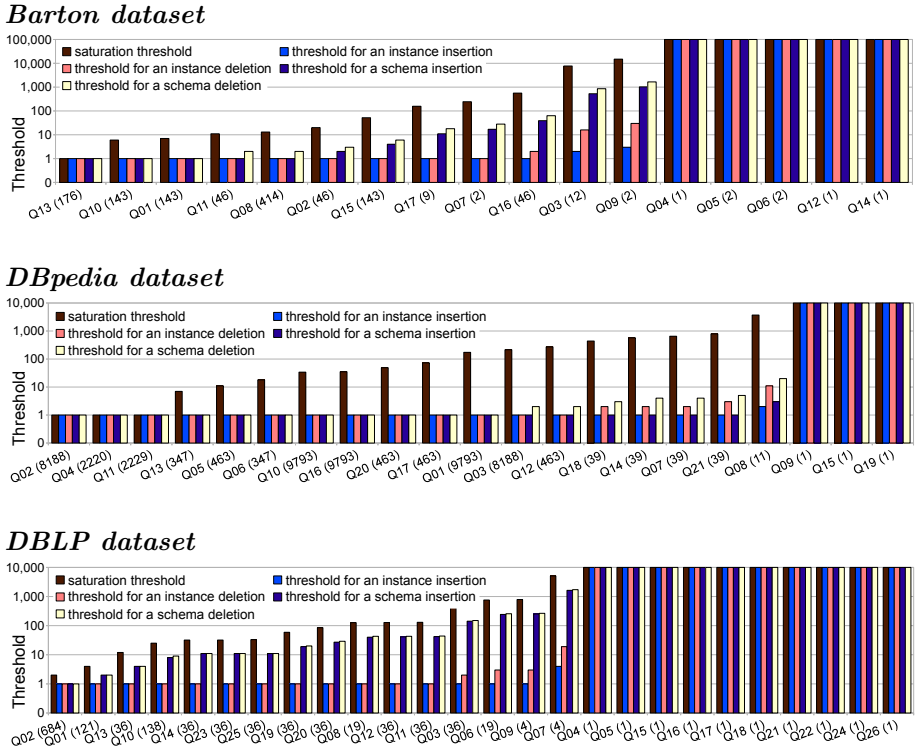


Figure 20: Saturation thresholds.

the reformulated query, the lower the threshold: saturation pays off faster when reformulation is expensive, and this tends to happen when the queries are syntactically complex. We see that $\mathbf{st}(q)$ varies from 1 (Q_{13}) to more than 10^4 (Q_{09}) on the Barton dataset, with similar results for the DBpedia and DBLP graphs. For queries such as Q_{04} (Barton), Q_{09} (DBpedia), Q_{04} (DBLP) etc., which are not affected by reformulation, the saturation cost can never be compensated. $\mathbf{st}$ is always higher than the update thresholds, which is normal since $\mathbf{st}$ runs offset *the complete saturation cost*, whereas $\mathbf{st}_\mathbf{i}^+$, $\mathbf{st}_\mathbf{i}^-$, $\mathbf{st}_\mathbf{s}^+$ and $\mathbf{st}_\mathbf{s}^-$ need to offset the cost of maintaining saturation for just one triple added or deleted. Finally, $\mathbf{st}_\mathbf{i}^+$ is lower than $\mathbf{st}_\mathbf{i}^-$, and $\mathbf{st}_\mathbf{s}^+$ is lower than $\mathbf{st}_\mathbf{s}^-$, meaning that saturation costs particularly penalize scenarios where deletions are frequent.

8.6 Conclusion of the experiments

Our experiments have shown that `Saturate` and `Reformulate` can be used to *process BGP queries* efficiently by exploiting an off-the-shelf RDBMS. However, they perform very differently depending on the query selectivity and the impact of the schema through reasoning: saturation is best for large-reformulation queries, while reformulation is efficient for small-to-moderate reformulation. With respect to *updates*, we have shown that saturation can be maintained at a reasonable cost for instance-level updates, while schema-level updates are much more expensive. Updates, however, have a small impact on reformulation making it appropriate for high update rates. When considering also *repeated query*

runs, we have highlighted a number of thresholds determining when saturation pays off; these thresholds are strongly impacted by the query reformulation size and selectivity. While saturation is the default in many RDF platforms, our experiments demonstrate the practical interest of *reformulation-based BGP query answering*.

9 Related work

Many well-known SPARQL compliant RDF management systems, e.g., 3store [23], Jena [26], OWLIM [27], Sesame [28], or Virtuoso [29], or research prototypes, e.g., Hexastore [22] or RDF-3X [17], either (*i*) ignore entailed triples or (*ii*) provide saturation-based query answering, based on (a subset of) RDF entailment rules.

The drawbacks of saturation w.r.t. updates have been pointed out in [7], which proposes a *truth maintenance* technique implemented in Sesame. It relies on the storage and management of the *justifications* of entailed triples (which triples beget them). While efficient on graphs with few entailed triples, the technique is pegged by the high overhead of handling justifications when their number and size grow. Therefore, [6] proposes to compute only the relevant justifications w.r.t. an update, at maintenance time. This technique is implemented in OWLIM, however [27] points out that schema-level deletions can lead to poor performance. In contrast, our saturation maintenance technique (Section 6.2) is based on the *number of times* triples are entailed; this is easier to store and manipulate. Our technique performs well for instance updates, while it has various behavior for schema updates. On average, it is worth maintaining the saturation, though in some cases it may be more costly than re-saturating (up to five times in our experiments). A distinct yet related problem studied in [13] is finding which triples to delete from a graph, so that an implicit triple no longer holds.

Reformulation-based query answering has been investigated in RDF fragments ranging from the Description Logic (DL) [5] one [3, 8, 11], i.e., modeling simple DL knowledge bases, to a slight extension thereof allowing values to be used both as constants and classes/properties [4, 10, 15, 19]. These two fragments of RDF impose restrictions on triples (no blank nodes) and on entailment (only the RDFS entailment rules are considered). Our DB fragment is *strictly more expressive* since it supports blank nodes.

Reformulation-based query answering in the DL fragment of RDF has been investigated for relational conjunctive queries [3, 8, 11], while the slight extension thereof considered in [4, 10, 15, 19] has been investigated for one-triple BGP queries [15, 19], BGP queries [10], and SPARQL queries [4]. Relational conjunctive queries are *strictly less expressive* than BGP queries, since the former only allow triples of the form `s` rdf:type c . and `s` p `o` ., ruling out the possibility to put variables in place of classes or properties.

The query reformulation algorithms of [3, 10] are restrictions of our `Reformulate`. The same holds for the algorithms in [8, 11] when restricted to the DL fragment of RDF, whereas they are capable of handling complex DLs. Algorithms in [3, 8, 11] consider only our rules (14)–(17) to reformulate relational conjunctive queries, while the algorithm in [10] needs two additional rules for BGP queries. These two rules actually correspond to our rules

(5)–(13), *under the simplifying assumption* that part of the information needed for reformulation have been pre-computed. In [15, 19], atomic BGP queries are reformulated using a standard backward-chaining algorithm [18] on first order encodings of the entailment rules dedicated to RDFS. In [4], SPARQL queries are reformulated into *nested* SPARQL, i.e., an extension of SPARQL in which properties in triples can be nested regular expressions. While such nested reformulated queries are more compact, the queries we produce are more practical, since their evaluation can be directly delegated to *any* off-the-shelf RDBMS, or to an RDF engine such as RDF-3X [17] even if it is unaware of reasoning.

10 Conclusion

In this paper, we have extended the state of the art in BGP query answering against RDF graphs with updates. We have devised novel saturation- and reformulation-based query answering techniques robust to instance and schema updates, for an RDF fragment extending those known in the literature; we compared thoroughly their performance and identified the factors impacting the comparison. Notably, our techniques can be directly deployed on top of any off-the-shelf RDBMS. An automated strategy to chose between the two techniques is part of our future work.

References

[1] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. Scalable semantic web data management using vertical partitioning. In *VLDB*, 2007.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] P. Adjiman, F. Goasdoué, and M.-C. Rousset. SomeRDFS in the semantic web. *JODS*, 8, 2007.

[4] M. Arenas, C. Gutierrez, and J. Pérez. Foundations of RDF databases. In *Reasoning Web*, 2009.

[5] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[6] B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, and R. Velkov. OWLIM: A family of scalable semantic repositories. *Semantic Web*, 2(1), 2011.

[7] J. Broekstra and A. Kampman. Inferencing and truth maintenance in RDF Schema: Exploring a naive practical approach. In *PSSS Workshop*, 2003.

[8] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *Journal of Automated Reasoning (JAR)*, 39(3), 2007.

[9] `http://kdl.cs.umass.edu/data/dblp/dblp-info.html`.

[10] F. Goasdoué, K. Karanasos, J. Leblay, and I. Manolescu. View selection in semantic web databases. *PVLDB*, 2011.

[11] G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization. In *ICDE*, 2011. Keynote.

[12] C. Gutierrez, C. Hurtado, and A. Vaisman. The meaning of erasing in RDF under the Katsuno-Mendelzon approach. In *WebDB*, 2006.

[13] C. Gutierrez, C. A. Hurtado, and A. A. Vaisman. RDFS update: From theory to practice. In *ESWC*, 2011.

[14] T. Imielinski and W. L. Jr. Incomplete information in relational databases. *JACM*, 31(4), 1984.

[15] Z. Kaoudi, I. Miliaraki, and M. Koubarakis. RDFS reasoning and query answering on DHTs. In *ISWC*, 2008.

[16] T. Neumann and G. Weikum. The RDF-3X engine for scalable management of RDF data. *VLDB J.*, 19(1), 2010.

[17] T. Neumann and G. Weikum. x-RDF-3X: Fast querying, high update rates, and consistency for RDF databases. *PVLDB*, 3(1), 2010.

[18] S. J. Russell and P. Norvig. *Artificial Intelligence - A Modern Approach*. Pearson Education, 2010.

[19] J. Urbani, F. van Harmelen, S. Schlobach, and H. Bal. QueryPIE: Backward reasoning for OWL Horst over very large knowledge bases. In *ISWC*, 2011.

[20] W3C. SPARQL protocol and RDF query language. `http://www.w3.org/TR/rdf-sparql-query`.

[21] Resource description framework. `http://www.w3.org/RDF`.

[22] C. Weiss, P. Karras, and A. Bernstein. Hexastore: sextuple indexing for Semantic Web data management. *PVLDB*, 1(1), 2008.

[23] 3store. `http://www.aktors.org/technologies/3store`.

[24] Barton. `http://simile.mit.edu/rdf-test-data/barton`.

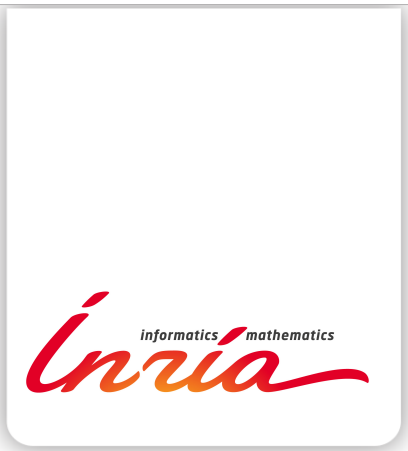
[25] DBpedia 3.7. `http://wiki.dbpedia.org/Downloads37`.

[26] Jena. `http://jena.sourceforge.net`.

[27] Owlim. `http://owlim.ontotext.com`.

[28] Sesame. `http://www.openrdf.org`.

[29] Virtuoso. `http://virtuoso.openlinksw.com`.



RESEARCH CENTRE
SACLAY – ÎLE-DE-FRANCE

Parc Orsay Université
4 rue Jacques Monod
91893 Orsay Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399