Learning Commonalities in RDF and SPARQL

Sara El Hassad, François Goasdoué, Hélène Jaudoin

Learning Commonalities in RDF and SPARQL

v4 of the Research Report made available online in early september 2016 (v1).

The main change in v2 was the addition of Section 5. In v3, some results and discussions were added to the core Sections 3 and 4; also details were added to the description of the related work in Section 7. In v4, experiments with DBpedia were added to Section 6.

Sara El Hassad
IRISA, Univ. Rennes 1
Lannion, France
sara.el-hassad@irisa.fr

François Goasdoué
IRISA, Univ. Rennes 1
Lannion, France
fg@irisa.fr

Hélène Jaudoin
IRISA, Univ. Rennes 1
Lannion, France
helene.jaudoin@irisa.fr

ABSTRACT

Finding commonalities between descriptions of data or knowledge is a fundamental task in Machine Learning. The formal notion characterizing precisely such commonalities is known as *least general generalization* of descriptions and was introduced by G. Plotkin in the early 70's, in First Order Logic.

Identifying least general generalizations has a large scope of database applications ranging from query optimization (e.g., to share commonalities between queries in view selection or multi-query optimization) to recommendation in social networks (e.g., to establish connections between users based on their commonalities between profiles or searches).

To the best of our knowledge, this is the first work that revisits the notion of least general generalizations in the entire Resource Description Framework (RDF) and popular conjunctive fragment of SPARQL, a.k.a. Basic Graph Pattern (BGP) queries. Our contributions include the definition and the computation of least general generalizations in these two settings, which amounts to finding the largest set of commonalities between incomplete databases and conjunctive queries, under deductive constraints. We also provide an experimental assessment of our technical contributions.

1. INTRODUCTION

Finding the commonalities between descriptions of data or knowledge is a foundational reasoning problem of Machine Learning, which was formalized in the early 70's as computing a *least general generalization* (`lgg`) of such descriptions [39, 40]. Since the early 90's, this problem has also received consideration in the Knowledge Representation field, where least general generalization were rebaptized *least common subsumers* [18], in Description Logics, e.g., [18, 10, 11, 50] and in Conceptual Graphs [17].

Interestingly, this problem has a large scope of applications when transposed in a database setting, i.e., when descriptions are expressed in a given data model or query language. In *Query Optimization*, `lgg`s of subsets of a query workload $\mathcal{W}$ may be used to identify candidate views or potential query sharing, with which $\mathcal{W}$ can be efficiently evaluated, in the View Selection problem or in the Multi-Query Optimization problem, respectively. In *Recommendation*, in particular in a social context, `lgg`s of sets of user descriptions (i.e., profiles) may help recommending users other users, or to form a community, when they share enough interests. Also, `lgg`s of sets of queries issued by distinct users may help recommending users to each other, if what they ask for is enough related. In *Exploration*, `lgg`s of datasets may be used to classify/categorize them w.r.t. their common information, to identify common social graph patterns between organizations (e.g., criminal ones), or to help identifying new potential links between datasets in the Linked Data Cloud.

Contributions. In this work, we revisit the problem of computing least general generalizations in the RDF data model and its SPARQL query language, the prominent Semantic Web standards of the W3C. This is a necessary first step towards using `lgg`s within central database problems. More precisely, our contributions to this problem are:

1. We define and study the problem of computing an `lgg` of RDF graphs in the *entire RDF standard*: we do not restrict RDF graphs in any way, i.e., neither their structure nor their semantics defined upon RDF entailment (inference). The recent work [20, 19] brings a limited solution to the problem. It allows finding the commonalities between *single entities* extracted from RDF graphs (e.g., users in a social network), *ignoring* RDF entailment. In contrast, we further aim at considering the problem in all its generality, i.e., finding the commonalities between *general* RDF graphs, hence modeling *multiple interrelated entities* (e.g., social networks of users), accurately w.r.t. their standard semantics.

2. We define and study the problem of computing an `lgg` of *general* SPARQL conjunctive queries, a.k.a. Basic Graph Pattern Queries (BGPQs), while the literature only considers unary tree-shaped conjunctive queries (UTCQ) [31]. Further, when available, we devise how to use background knowledge, formalized as ontological constraints modeling the application domain, in order to compute much more pregnant `lgg`s.

3. We provide algorithms for our solutions to these two

problems. In particular, since RDF graphs may be large, we provide algorithms that allow computing `lgg`s of *small-to-huge general RDF graphs* (i.e., that fit either in memory, in data management systems or in MapReduce clusters) w.r.t. *any set of entailment rules* from the RDF standard.

4. We provide experiments using *synthetic* LUBM data and *real* DBpedia data, which show to which extent `lgg`s of BGPQs are much more pregnant when background knowledge is available.

The paper is organized as follows. In Section 2, we introduce the RDF data model and its SPARQL query language. Then, we study the problem of computing an `lgg` of RDF graphs in Section 3, and of BGPQs in Section 4. In Section 5, we present algorithms for the problems studied in the two preceding Sections. We report on the experiments we conducted in Section 6. Finally, we present related works in Section 7 and we conclude and draw some perspectives in Section 8.

2. PRELIMINARIES

We introduce the *RDF data model* in Section 2.1. Then, in Section 2.2, we present the SPARQL conjunctive queries, a.k.a. *Basic Graph Pattern queries (BGPQs)*.

2.1 Resource Description Framework (RDF)

2.1.1 RDF graphs

The RDF data model allows specifying *RDF graphs*. An RDF graph is a set of *triples* of the form $(\mathtt{s}, \mathtt{p}, \mathtt{o})$. A triple states that its *subject* `s` has the *property* `p`, the value of which is the *object* `o`. Triples are built using three pairwise disjoint sets: a set $\mathcal{U}$ of *uniform resources identifiers (URIs)*, a set $\mathcal{L}$ of *literals* (constants), and a set $\mathcal{B}$ of *blank nodes* allowing to support *incomplete information*. Blank nodes are local identifiers for missing values within an RDF graph (unknown URIs or literals); they are the RDF counterparts of labelled nulls or existential variables in incomplete relational databases [28, 9], as shown in [25]. *Well-formed triples*, as per the RDF specification [46], belong to $(\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{L} \cup \mathcal{B})$; we only consider such triples hereafter.

Notations. We use `s`, `p`, `o` in triples as placeholders. We note $\mathtt{Val}(\mathcal{G})$ the set of *values* occurring in an RDF graph $\mathcal{G}$, i.e., the URIs, literals and blank nodes; we note $\mathtt{Bl}(\mathcal{G})$ the set of blank nodes occurring in $\mathcal{G}$. A blank node is written b possibly with a subscript, and a literal is a string between quotes. For instance, the triples $(b, \text{hasTitle}, \text{“LGG in RDF”})$ and $(b, \text{hasContactAuthor}, b_1)$ state that *something (b) entitled “LGG in RDF” has somebody (b_1) as contact author*.

A triple models an assertion, either for a *class* (unary relation) or for a *property* (binary relation). Table 1 (top) shows the use of triples to state such assertions, as well as the relational assertions to which they correspond. The RDF standard [46] provides built-in classes and properties, as URIs within the `rdf` and `rdfs` pre-defined namespaces, e.g., rdf:type which can be used to state that the above b is a conference paper with the triple $(b, \text{rdf:type}, \text{ConfPaper})$.

2.1.2 Adding ontological knowledge to RDF graphs

RDF statement	Triple	Relational assertion
Class assertion	$(\mathtt{s}, \text{rdf:type}, \mathtt{o})$	$\mathtt{o}(\mathtt{s})$
Property assertion	$(\mathtt{s}, \mathtt{p}, \mathtt{o})$ with $\mathtt{p} \neq \text{rdf:type}$	$\mathtt{p}(\mathtt{s}, \mathtt{o})$

RDFS statement	Triple	Relational constraint
Subclass	$(\mathtt{s}, \text{rdfs:subClassOf}, \mathtt{o})$	$\mathtt{s} \subseteq \mathtt{o}$
Subproperty	$(\mathtt{s}, \text{rdfs:subPropertyOf}, \mathtt{o})$	$\mathtt{s} \subseteq \mathtt{o}$
Domain typing	$(\mathtt{s}, \text{rdfs:domain}, \mathtt{o})$	$\Pi_{\text{domain}}(\mathtt{s}) \subseteq \mathtt{o}$
Range typing	$(\mathtt{s}, \text{rdfs:range}, \mathtt{o})$	$\Pi_{\text{range}}(\mathtt{s}) \subseteq \mathtt{o}$

Table 1: RDF (top) & RDFS (bottom) statements.

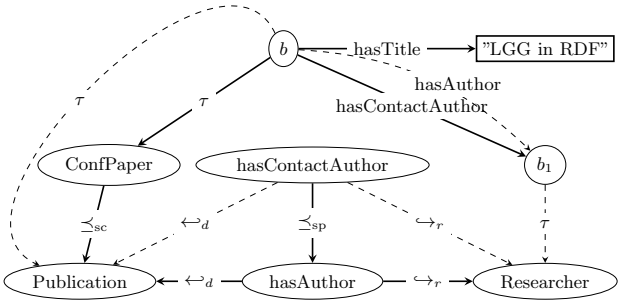

Figure 1: Sample RDF graph $\mathcal{G}$.

An essential feature of RDF is the possibility to enhance the descriptions in RDF graphs by declaring *ontological constraints* between the classes and properties they use. This is achieved with *RDF Schema (RDFS)* statements, which are triples using particular buit-in properties. Table 1 (bottom) lists the allowed constraints, the triples to state them, as well as the *deductive* relational constraints to which they correspond; *domain* and *range* denote respectively the first and second attribute of every property. For example, the triple $(\text{ConfPaper}, \text{rdfs:subClassOf}, \text{Publication})$ states that *conference papers are publications*, the triple $(\text{hasContactAuthor}, \text{rdfs:subPropertyOf}, \text{hasAuthor})$ states that *having a contact author is having an author*, the triple $(\text{hasAuthor}, \text{rdfs:domain}, \text{Publication})$ states that *only publications have authors* while the triple $(\text{hasAuthor}, \text{rdfs:range}, \text{Researcher})$ states that *only researchers are authors of something*.

Notations. From now, for conciseness, we use the following shorthands for built-in properties: τ for rdf:type, $\prec_{\text{sc}}$ for rdfs:subClassOf, $\prec_{\text{sp}}$ for rdfs:subPropertyOf, $\hookleftarrow_d$ for rdfs:domain, and $\hookrightarrow_r$ for rdfs:range.

Figure 1 displays the usual representation of the RDF graph $\mathcal{G}$ made of the seven above-mentioned triples, which are called the *explicit triples* of $\mathcal{G}$. A triple $(\mathtt{s}, \mathtt{p}, \mathtt{o})$ corresponds to the `p`-labeled directed edge from the `s` node to the `o` node. Explicit triples are shown as solid edges, while the *implicit ones*, which are derived using ontological constraints (see below), are shown as dashed edges.

Importantly, the semantics of ontological constraints differ from the typical relational database setting, where they are interpreted under the *closed world assumption (CWA)*, i.e., as *integrity constraints*. In RDF, they are interpreted under the *open world assumption (OWA)*, i.e., as *deductive constraints*. Under OWA, *all* the triples that can be deduced from the constraints, called *implicit triples*, are assumed to be part of the RDF graph even though they are not explicitly present in it. For instance, consider the RDF graph $\mathcal{G}$ in Figure 1. The ontological constraint $(\text{hasContactAuthor}, \text{rdfs:subPropertyOf}, \text{hasAuthor})$ is violated under CWA, as

Rule name [47]	Entailment rule
rdfs2	$(\mathtt{p}, \hookrightarrow_d, \mathtt{o}), (\mathtt{s}_1, \mathtt{p}, \mathtt{o}_1) \rightarrow (\mathtt{s}_1, \tau, \mathtt{o})$
rdfs3	$(\mathtt{p}, \hookrightarrow_r, \mathtt{o}), (\mathtt{s}_1, \mathtt{p}, \mathtt{o}_1) \rightarrow (\mathtt{o}_1, \tau, \mathtt{o})$
rdfs5	$(\mathtt{p}_1, \preceq_{\text{sp}}, \mathtt{p}_2), (\mathtt{p}_2, \preceq_{\text{sp}}, \mathtt{p}_3) \rightarrow (\mathtt{p}_1, \preceq_{\text{sp}}, \mathtt{p}_3)$
rdfs7	$(\mathtt{p}_1, \preceq_{\text{sp}}, \mathtt{p}_2), (\mathtt{s}, \mathtt{p}_1, \mathtt{o}) \rightarrow (\mathtt{s}, \mathtt{p}_2, \mathtt{o})$
rdfs9	$(\mathtt{s}, \preceq_{\text{sc}}, \mathtt{o}), (\mathtt{s}_1, \tau, \mathtt{s}) \rightarrow (\mathtt{s}_1, \tau, \mathtt{o})$
rdfs11	$(\mathtt{s}, \preceq_{\text{sc}}, \mathtt{o}), (\mathtt{o}, \preceq_{\text{sc}}, \mathtt{o}_1) \rightarrow (\mathtt{s}, \preceq_{\text{sc}}, \mathtt{o}_1)$
ext1	$(\mathtt{p}, \hookrightarrow_d, \mathtt{o}), (\mathtt{o}, \preceq_{\text{sc}}, \mathtt{o}_1) \rightarrow (\mathtt{p}, \hookrightarrow_d, \mathtt{o}_1)$
ext2	$(\mathtt{p}, \hookrightarrow_r, \mathtt{o}), (\mathtt{o}, \preceq_{\text{sc}}, \mathtt{o}_1) \rightarrow (\mathtt{p}, \hookrightarrow_r, \mathtt{o}_1)$
ext3	$(\mathtt{p}, \preceq_{\text{sp}}, \mathtt{p}_1), (\mathtt{p}_1, \hookrightarrow_d, \mathtt{o}) \rightarrow (\mathtt{p}, \hookrightarrow_d, \mathtt{o})$
ext4	$(\mathtt{p}, \preceq_{\text{sp}}, \mathtt{p}_1), (\mathtt{p}_1, \hookrightarrow_r, \mathtt{o}) \rightarrow (\mathtt{p}, \hookrightarrow_r, \mathtt{o})$

Table 2: Sample set of RDF entailment rules.

the triple $(b, \text{hasContactAuthor}, b_1)$ is explicit in $\mathcal{G}$ while $(b, \text{hasAuthor}, b_1)$ is not. By contrast, under OWA, the same constraint allows deriving the "missing" triple $(b, \text{hasAuthor}, b_1)$, making it implicit within $\mathcal{G}$. Further, this implicit triple together with the explicit ontological constraint (hasAuthor, rdfs:range, Researcher) yields the implicit triple (b_1, rdf:type, Researcher).

2.1.3 Deriving the implicit triples of an RDF graph

The RDF standard defines a set of *entailment rules* in order to derive automatically *all* the triples that are implicit to an RDF graph. Table 2 shows the strict subset of these rules that we will use to illustrate important notions as well as our contributions in the next sections; importantly, our contributions hold for the whole set of entailment rules of the RDF standard, and any subset of thereof. The rules in Table 2 concern the derivation of implicit triples using ontological constraints (i.e., *RDFS statements*). They encode the *propagation* of assertions through constraints (`rdfs2`, `rdfs3`, `rdfs7`, `rdfs9`), the *transitivity* of the $\preceq_{\text{sp}}$ and $\preceq_{\text{sc}}$ constraints (`rdfs5`, `rdfs11`), the *complementation* of domains or ranges through $\preceq_{\text{sc}}$ (`ext1`, `ext2`), and the *inheritance* of domains and of ranges through $\preceq_{\text{sp}}$ (`ext3`, `ext4`).

The *saturation (a.k.a. closure)* of an RDF graph $\mathcal{G}$ w.r.t. a set $\mathcal{R}$ of RDF entailment rules, is the RDF graph $\mathcal{G}^\infty$ obtained by adding to $\mathcal{G}$ all the implicit triples that can be derived from $\mathcal{G}$ using $\mathcal{R}$. Roughly speaking, the saturation $\mathcal{G}^\infty$ *materializes* the semantics of $\mathcal{G}$. It corresponds to the fixpoint obtained by repeatedly applying the rules in $\mathcal{R}$ to $\mathcal{G}$ in a forward-chaining fashion, a.k.a. *chasing* $\mathcal{G}$ with $\mathcal{R}$ in the database parlance. In RDF, the saturation is *always* finite and unique (up to blank node renaming), and does not contain implicit triples [46, 47].

The saturation of the RDF graph $\mathcal{G}$ shown in Figure 1 corresponds to the RDF graph $\mathcal{G}^\infty$ in which all the $\mathcal{G}$ implicit triples have been made explicit. It is worth noting how, starting from $\mathcal{G}$, applying RDF entailment rules *mechanizes* the construction of $\mathcal{G}^\infty$. For instance, recall the reasoning sketched in Section 2.1.2 for deriving the triple (b_1, rdf:type, Researcher). This is automated by the following sequence of applications of RDF entailment rules: (hasContactAuthor, rdfs:subPropertyOf, hasAuthor) and $(b, \text{hasContactAuthor}, b_1)$ trigger the `rdfs7` rule which adds $(b, \text{hasAuthor}, b_1)$ to the RDF graph. In turn, this new triple together with (hasAuthor, rdfs:range, Researcher) triggers the `rdfs3` rule which adds (b_1, rdf:type, Researcher).

2.1.4 Comparing RDF graphs

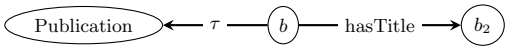

Figure 2: Sample RDF graph $\mathcal{G}'$.

The RDF standard defines a generalization/specialization relationship between two RDF graphs, called *entailment between graphs*. Roughly speaking, an RDF graph $\mathcal{G}$ is more specific than another RDF graph $\mathcal{G}'$, or equivalently $\mathcal{G}'$ is more general than $\mathcal{G}$, whenever there is an embedding of $\mathcal{G}'$ into the *saturation* of $\mathcal{G}$, i.e., the complete set of triples that $\mathcal{G}$ models.

More formally, given any subset $\mathcal{R}$ of RDF entailment rules, an RDF graph $\mathcal{G}$ *entails* an RDF graph $\mathcal{G}'$, denoted $\mathcal{G} \models_{\mathcal{R}} \mathcal{G}'$, iff there exists an homomorphism ϕ from $\mathtt{Bl}(\mathcal{G}')$ to $\mathtt{Val}(\mathcal{G}^\infty)$ such that $[\mathcal{G}']_\phi \subseteq \mathcal{G}^\infty$, where $[\mathcal{G}']_\phi$ is the RDF graph obtained from $\mathcal{G}'$ by replacing every blank node b by its image $\phi(b)$.

Figure 2 shows an RDF graph $\mathcal{G}'$ that is entailed by the RDF graph $\mathcal{G}$ in Figure 1 w.r.t. the RDF entailment rules displayed in Table 2. In particular, $\mathcal{G} \models_{\mathcal{R}} \mathcal{G}'$ holds for the homomorphism ϕ such that: $\phi(b) = b$ and $\phi(b_2) =$ "LGG in RDF". By contrast, when $\mathcal{R}$ is empty, this is not the case (i.e., $\mathcal{G} \not\models_{\mathcal{R}} \mathcal{G}'$), as the dashed edges in $\mathcal{G}$ are not materialized by saturation, hence the $\mathcal{G}'$ triple $(b, \tau, \text{Publication})$ cannot have an image in $\mathcal{G}$ through some homomorphism.

Notations. When relevant to the discussion, we designate by $\mathcal{G} \models^{\phi}_{\mathcal{R}} \mathcal{G}'$ the fact that the entailment $\mathcal{G} \models_{\mathcal{R}} \mathcal{G}'$ holds due to the graph homomorphism ϕ. Also, when RDF entailment rules are disregarded, i.e., $\mathcal{R} = \emptyset$, we note the entailment relation $\models$ (without indicating the rule set under consideration). Observe that, in this particular case, the entailment between RDF graphs collapses to the database notion of *containment* between two incomplete instances, each stored into a $\mathtt{Triple}(\mathtt{s}, \mathtt{p}, \mathtt{o})$ relation.

Importantly, some remarkable properties follow directly from the definition of entailment between two RDF graphs:

1. $\mathcal{G}$ and $\mathcal{G}^\infty$ are equivalent, noted $\mathcal{G} \equiv_{\mathcal{R}} \mathcal{G}^\infty$, since clearly $\mathcal{G} \models_{\mathcal{R}} \mathcal{G}^\infty$ and $\mathcal{G}^\infty \models_{\mathcal{R}} \mathcal{G}$ hold,
2. $\mathcal{G} \models_{\mathcal{R}} \mathcal{G}'$ iff $\mathcal{G}^\infty \models \mathcal{G}'$ holds.

In particular, the second above property points out how entailment reduces to standard database containment once saturation is computed.

2.2 SPARQL conjunctive queries

2.2.1 Basic graph pattern queries

The well-established conjunctive fragment of SPARQL queries, a.k.a. *basic graph pattern queries (BGPQs)*, is the counterpart of the relational select-project-join queries; it is the most widely used subset of SPARQL queries in real-world applications [36].

A *Basic Graph Pattern (BGP)* is a set of *triple patterns*, or simply triples by a slight abuse of language. They generalize RDF triples by allowing the use of variables. Given a set $\mathcal{V}$ of variables, pairwise disjoint with $\mathcal{U}$, $\mathcal{L}$ and $\mathcal{B}$, triple patterns belong to: $(\mathcal{V} \cup \mathcal{U} \cup \mathcal{B}) \times (\mathcal{V} \cup \mathcal{U}) \times (\mathcal{V} \cup \mathcal{U} \cup \mathcal{L} \cup \mathcal{B})$.

Notations. We adopt the usual conjunctive query notation $q(\bar{x}) \leftarrow t_1, \ldots, t_\alpha$, where $\{t_1, \ldots, t_\alpha\}$ is a BGP; the query head variables $\bar{x}$ are called *answer variables*, and form a subset of the variables occurring in $t_1, \ldots, t_\alpha$; for boolean

queries, $\bar{x}$ is empty. The head of q, noted $head(q)$, is $q(\bar{x})$, and the body of q, denoted $body(q)$, is the BGP $\{t_1, \ldots, t_\alpha\}$. We use x and y in queries, possibly with subscripts, for answer and non-answer variables respectively. Finally, we denote by $\texttt{VarBl}(q)$ the set of variables *and* blank nodes occurring in the query q; we note $\texttt{Val}(q)$ the set of all its values, i.e., URIs, blank nodes, literals and variables.

2.2.2 Entailing and answering queries

Two important notions characterize how an RDF graph contributes to a query.

Query entailment. The weaker notion indicates whether or not an RDF graph holds some answer(s) to a query. It generalizes entailment between RDF graphs, to account for the presence of variables in the query body, for establishing whether a graph entails a query, i.e., whether the query embeds in that graph.

Formally, given a BGPQ q, an RDF graph $\mathcal{G}$ and a set $\mathcal{R}$ of RDF entailment rules, $\mathcal{G}$ *entails* q, denoted $\mathcal{G} \models_{\mathcal{R}} q$, iff $\mathcal{G} \models_{\mathcal{R}} body(q)$ holds, i.e., there exists a homomorphism ϕ from $\texttt{VarBl}(q)$ to $\texttt{Val}(\mathcal{G}^\infty)$ such that $[body(q)]_\phi \subseteq \mathcal{G}^\infty$.

The RDF graph $\mathcal{G}$ in Figure 1 entails the query $q(x_1, x_2) \leftarrow (x_1, \tau, x_2)$ asking for all the resources and their classes, for instance because of the homomorphism ϕ such that $\phi(x_1) = b$ and $\phi(x_2) = \text{ConfPaper}$. Observe that this entailment holds for any subset of RDF entailment rules, since the above ϕ already holds for $\mathcal{R} = \emptyset$, i.e., considering only the explicit triples in Figure 1.

Notations. Similarly to RDF graph entailment, we denote by $\mathcal{G} \models^\phi_{\mathcal{R}} q$ that the entailment $\mathcal{G} \models_{\mathcal{R}} q$ holds due to the homomorphism ϕ.

Query answering. The stronger notion characterizing how an RDF graph contributes to a query, which builds on the preceding one, identifies *all* the query answers that the graph holds. Formally, assuming the head of a BGPQ q is $q(\bar{x})$, the *answer set of* q *against* $\mathcal{G}$ is:

$$q(\mathcal{G}) = \{(\bar{x})_\phi \mid \mathcal{G} \models^\phi_{\mathcal{R}} body(q)\}$$

where we denote by $(\bar{x})_\phi$ the tuple of $\mathcal{G}^\infty$ values obtained by replacing every answer variable $x_i \in \bar{x}$ by its image $\phi(x_i)$. Observe that when $\mathcal{R}$ is empty, BGPQ query answering coincide with *standard relational conjunctive query evaluation.*

The answer set to the above query $q(x_1, x_2) \leftarrow (x_1, \tau, x_2)$ against the RDF graph $\mathcal{G}$ in Figure 1 is:

- $\{(b, \text{ConfPaper}), (b, \text{Publication}), (b_1, \text{Researcher})\}$ for $\mathcal{R}$ the set of entailment rules shown in Table 2, i.e., considering the explicit *and* implicit triples in Figure 1,
- $\{(b, \text{ConfPaper})\}$ for $\mathcal{R} = \emptyset$, i.e., considering only the explicit triples in Figure 1.

Importantly, from the definition of entailment between two RDF graphs [46, 47], the following holds:

PROPERTY 1. *Given two RDF graphs $\mathcal{G}, \mathcal{G}'$ and a set $\mathcal{R}$ of RDF entailment rules, (i) $\mathcal{G}$ and $\mathcal{G}^\infty$ are equivalent ($\mathcal{G} \models_{\mathcal{R}} \mathcal{G}^\infty$ and $\mathcal{G}^\infty \models_{\mathcal{R}} \mathcal{G}$ hold), noted $\mathcal{G} \equiv_{\mathcal{R}} \mathcal{G}^\infty$, and (ii) $\mathcal{G} \models_{\mathcal{R}} \mathcal{G}'$ holds iff $\mathcal{G}^\infty \models \mathcal{G}'$ holds.*

From a practical viewpoint, Property 1 points out that checking $\mathcal{G} \models_{\mathcal{R}} \mathcal{G}'$ can be done in two steps: a reasoning step that computes the saturation $\mathcal{G}^\infty$ of $\mathcal{G}$, followed by a standard graph homomorphism step that checks if $\mathcal{G}^\infty \models \mathcal{G}'$ holds.

2.2.3 Comparing queries

Similarly to RDF graphs, queries can be compared through the generalization/specialization relationship of *entailment between queries*. This relationship is the counterpart of that *query containment* in the relational database setting.

Let q, q' be two BGPQs with *same* arity, whose heads are $q(\bar{x})$ and $q'(\bar{x}')$, and $\mathcal{R}$ the set of RDF entailment rules under consideration. q *entails* q', denoted $q \models_{\mathcal{R}} q'$, iff $body(q) \models^\phi_{\mathcal{R}} body(q')$ *with* $(\bar{x}')_\phi = \bar{x}$ holds. Here, $body(q) \models^\phi_{\mathcal{R}} body(q')$ is the adaptation of the above-mentioned entailment relationships between RDF graphs to the fact that the query bodies may feature variables, i.e., ϕ is a homomorphism from $\texttt{VarBl}(q')$ to $\texttt{Val}(q)$ such that $[body(q')]_\phi \subseteq body(q)^\infty$; the saturation of a BGP, here $body(q)^\infty$, is the generalization of RDF graph saturation that treats variables as blank nodes, since they both equivalently model unknown information within BGPs [46, 47].

For instance, the query $q_1(x)$:- $(x, \tau, \text{ConfPaper})$, $(x, \text{hasContactAuthor}, y)$ entails the query $q_2(x)$:- (x, τ, y) with $\phi(x) = x$, $\phi(y) = \text{ConfPaper}$ and any entailment rule set. However, it is worth noting that, *counter-intuitively*, q_1 does not entail the query $q_3(x)$:- $(x, \tau, \text{Publication}), (x, \text{hasAuthor}, y)$. The reason is that query entailment does not consider extra ontological constraints, which would be needed by RDF entailment rules to find such elaborate entailment (here, that conference papers are publications, and that having a contact author is having an author). Extending standard entailment between BGPQs to take into account *extra ontological constraints* modeling background knowledge, is one contribution in this work (Section 4).

Remark that entailment between queries, query entailment and query answering (obviously) relate as follows:

PROPERTY 2. *Given an RDF graph $\mathcal{G}$, a set $\mathcal{R}$ of entailment rules and two BGPQs q, q' such that $q \models_{\mathcal{R}} q'$, (i) if $\mathcal{G} \models_{\mathcal{R}} q$ holds then $\mathcal{G} \models_{\mathcal{R}} q'$ holds, and (ii) $q(\mathcal{G}) \subseteq q'(\mathcal{G})$ holds.*

Finally, remark that query entailment, query answering and entailment between queries *treat the blank nodes in a query exactly as non-answer variables* [48]. Therefore, from now on, we assume *without loss of generality* that queries do not use blank nodes.

3. FINDING COMMONALITIES BETWEEN RDF GRAPHS

In Section 3.1, we first define the largest set of commonalities between RDF graphs as a particular RDF graph representing their *least general generalization* (`lgg` for short). Then, we devise a technique for computing such an `lgg` in Section 3.2.

3.1 Defining the `lgg` of RDF graphs

A *least general generalization* of n descriptions $d_1, \ldots, d_n$ is a most specific description d generalizing every $d_{1 \leq i \leq n}$ for some generalization/specialization relation between descriptions [39, 40]. In our RDF setting, we use RDF graphs as descriptions and entailment between RDF graphs as relation for generalization/specialization:

DEFINITION 1 (`lgg` OF RDF GRAPHS). *Let $\mathcal{G}_1, \ldots, \mathcal{G}_n$ be RDF graphs and $\mathcal{R}$ a set of RDF entailment rules.*

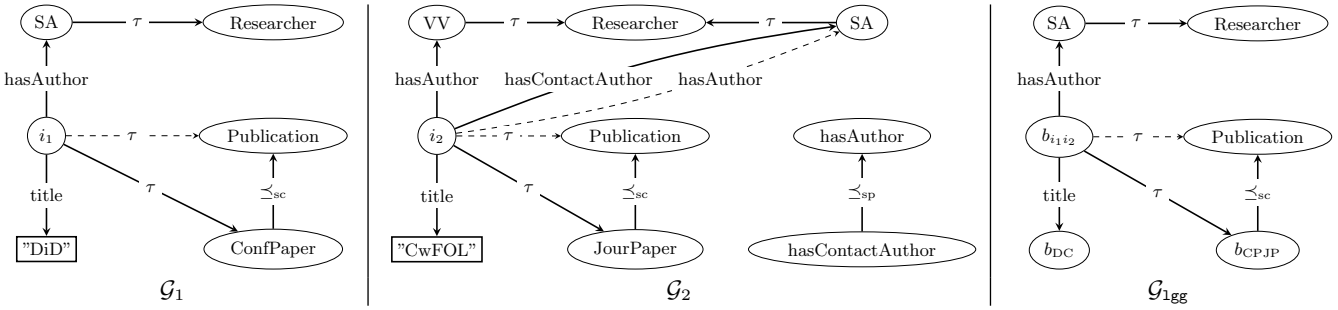

Figure 3: Sample RDF graphs $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{G}_{\texttt{lgg}}$, with $\mathcal{G}_{\texttt{lgg}}$ the minimal lgg of $\mathcal{G}_1$ and $\mathcal{G}_2$; their implicit triples (i.e., derived by the rules in Table 2) are shown as dashed edges.

- *A* generalization *of* $\mathcal{G}_1, \ldots, \mathcal{G}_n$ *is an RDF graph* $\mathcal{G}_g$ *such that* $\mathcal{G}_i \models_{\mathcal{R}} \mathcal{G}_g$ *holds for* $1 \leq i \leq n$.
- *A* least general generalization *(*lgg*) of* $\mathcal{G}_1, \ldots, \mathcal{G}_n$ *is a generalization* $\mathcal{G}_{\texttt{lgg}}$ *of* $\mathcal{G}_1, \ldots, \mathcal{G}_n$ *such that for any other generalization* $\mathcal{G}_g$ *of* $\mathcal{G}_1, \ldots, \mathcal{G}_n$, $\mathcal{G}_{\texttt{lgg}} \models_{\mathcal{R}} \mathcal{G}_g$ *holds.*

Importantly, in the RDF setting, the following holds:

THEOREM 1. *An* lgg *of RDF graphs always exists, and is* unique *up to entailment.*

PROOF. An lgg of RDF graphs always exists, since we can always construct a (possibly empty) RDF graph that is the lgg of RDF graphs, in particular *the cover graph of RDF graphs* devised in Section 3.2.

Also, an lgg of RDF graphs is *unique* up to entailment (since $\mathcal{G}_{\texttt{lgg}} \models_{\mathcal{R}} \mathcal{G}_g$ holds for any $\mathcal{G}_g$ in Definition 1). Indeed, if it were that RDF graphs have *multiple* lggs *incomparable* w.r.t. entailment, say $\texttt{lgg}_1$, ..., $\texttt{lgg}_m$, their merge[1] $\texttt{lgg}_1 \uplus \cdots \uplus \texttt{lgg}_m$ would be a *single strictly more specific* lgg, a contradiction. □

Figure 3 displays two RDF graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, as well as their minimal lgg (with lowest number of triples) when we consider the RDF entailment rules shown in Table 2: $\mathcal{G}_{\texttt{lgg}}$. $\mathcal{G}_1$ describes a conference paper i_1 with title "Disaggregations in Databases" and author Serge Abiteboul, who is a researcher; also conference papers are publications. $\mathcal{G}_2$ describes a journal paper i_2 with title "Computing with First-Order Logic", contact author Serge Abiteboul and author Victor Vianu, who are researchers; moreover, journal papers are publications and having a contact author is having an author. $\mathcal{G}_{\texttt{lgg}}$ states that their common information comprises the existence of a resource ($b_{i_1 i_2}$) having some type ($b_{\text{C(onf)P(aper)J(our)P(aper)}}$), which is a particular case of publication, with some title ($b_{\text{D(iD)C(wFOL)}}$) and author Serge Abiteboul, who is a researcher.

Though unique up to entailment (i.e., semantically unique), an lgg may have many syntactical forms due to *redundant* triples. Such triples can be either explicit ones that could have been left implicit if the set of RDF entailment rules under consideration allows deriving them from the remaining triples (e.g., materializing the only $\mathcal{G}_{\texttt{lgg}}$ implicit triple in Figure 3 would make it redundant if we consider the RDF entailment rules in Table 2) or triples generalizing others without needing RDF entailment rules, i.e., in a purely relational fashion (e.g., adding the triple $(b, \text{hasAuthor}, b')$ to $\mathcal{G}_{\texttt{lgg}}$ in Figure 3 would be redundant w.r.t. $(b_{i_1 i_2}, \text{hasAuthor}, SA)$). Also, an lgg may have *several minimal* syntactical variants obtained by pruning out redundant triples. For example, think of a minimal lgg comprising the triples $(A, \preceq_{\text{sc}}, B)$, $(B, \preceq_{\text{sc}}, A)$ and (b, τ, A), i.e., there exists an instance of the class A, which is equivalent to class B. Clearly, an equivalent and minimal variant of this lgg is the RDF graph comprising the triples $(A, \preceq_{\text{sc}}, B)$, $(B, \preceq_{\text{sc}}, A)$ and (b, τ, B).

Importantly, the above discussion is not specific to lggs of RDF graphs, since any RDF graph may feature redundancy. The detection and elimination of RDF graph redundancy has been studied in the literature, e.g., [33, 37, 38], hence we focus in this work on finding *some* lgg of RDF graphs.

The proposition below shows that an lgg of n RDF graphs, with $n \geq 3$, can be defined (hence computed) as a sequence of $n-1$ lggs of *two* RDF graphs. Intuitively, assuming that $\ell_{k \geq 2}$ is an operator computing an lgg of k input RDF graphs, the next proposition establishes that:

$$\begin{array}{ll} \ell_3(\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3) & \equiv_{\mathcal{R}} \ \ell_2(\ell_2(\mathcal{G}_1, \mathcal{G}_2), \mathcal{G}_3) \\ \ldots & \ldots \\ \ell_n(\mathcal{G}_1, \ldots, \mathcal{G}_n) & \equiv_{\mathcal{R}} \ \ell_2(\ell_{n-1}(\mathcal{G}_1, \ldots, \mathcal{G}_{n-1}), \mathcal{G}_n) \\ & \equiv_{\mathcal{R}} \ \ell_2(\ell_2(\cdots \ell_2(\ell_2(\mathcal{G}_1, \mathcal{G}_2), \mathcal{G}_3) \cdots, \mathcal{G}_{n-1}), \mathcal{G}_n) \end{array}$$

PROPOSITION 1. *Let* $\mathcal{G}_1, \ldots, \mathcal{G}_{n \geq 3}$ *be* n *RDF graphs and* $\mathcal{R}$ *a set of RDF entailment rules.* $\mathcal{G}_{\texttt{lgg}}$ *is an* lgg *of* $\mathcal{G}_1, \ldots, \mathcal{G}_n$ *iff* $\mathcal{G}_{\texttt{lgg}}$ *is an* lgg *of an* lgg *of* $\mathcal{G}_1, \ldots, \mathcal{G}_{n-1}$ *and* $\mathcal{G}_n$.

PROOF. The proof relies on the next lemma.

LEMMA 1. *Let* $\mathcal{G}_1, \ldots, \mathcal{G}_n$ *be RDF graphs and* $\mathcal{R}$ *a set of RDF entailment rules. If* $\mathcal{G}^1_{\texttt{lgg}}$ *is an* lgg *of* $\mathcal{G}_1, \ldots, \mathcal{G}_{k<n}$ *and* $\mathcal{G}^2_{\texttt{lgg}}$ *is an* lgg *of* $\mathcal{G}_1, \ldots, \mathcal{G}_n$, *then* $\mathcal{G}^1_{\texttt{lgg}} \models_{\mathcal{R}} \mathcal{G}^2_{\texttt{lgg}}$ *holds.*

Let us show that the above lemma holds.

Suppose that $\mathcal{G}^1_{\texttt{lgg}}$ is an lgg of $\mathcal{G}_1, \ldots, \mathcal{G}_{k<n}$, i.e., by Definition 1: (*i*) $\mathcal{G}_{1 \leq i \leq k} \models_{\mathcal{R}} \mathcal{G}^1_{\texttt{lgg}}$ holds and (*ii*) for any RDF graph $\mathcal{G}$ such that $\mathcal{G}_{1 \leq i \leq k} \models_{\mathcal{R}} \mathcal{G}$ holds, $\mathcal{G}^1_{\texttt{lgg}} \models_{\mathcal{R}} \mathcal{G}$ holds.

Suppose also that $\mathcal{G}^2_{\texttt{lgg}}$ is an lgg of $\mathcal{G}_1, \ldots, \mathcal{G}_n$, i.e., by Definition 1: (*i*) $\mathcal{G}_{1 \leq i \leq n} \models_{\mathcal{R}} \mathcal{G}^2_{\texttt{lgg}}$ holds and (*ii*) for any RDF graph $\mathcal{G}'$ such that $\mathcal{G}_{1 \leq i \leq n} \models_{\mathcal{R}} \mathcal{G}'$ holds, $\mathcal{G}^2_{\texttt{lgg}} \models_{\mathcal{R}} \mathcal{G}'$ holds.

[1]The merge of RDF graphs, performed with the RDF specific merge operator $\uplus$, is an RDF graph comprising the union of the input RDF graph *after renaming* their blank nodes with fresh ones, so that these RDF graphs do not share (thus join on) such values. Indeed, blank nodes are used to characterize the incompleteness of an RDF graph, hence are *local* to it (Section 2).

Clearly, $\mathcal{G}^2_{\texttt{lgg}}$ is a possible value for $\mathcal{G}$ above, because $\mathcal{G}_{1\leq i\leq n} \models_{\mathcal{R}} \mathcal{G}^2_{\texttt{lgg}}$ implies $\mathcal{G}_{1\leq i\leq k<n} \models_{\mathcal{R}} \mathcal{G}^2_{\texttt{lgg}}$, hence $\mathcal{G}^1_{\texttt{lgg}} \models_{\mathcal{R}} \mathcal{G}^2_{\texttt{lgg}}$ must hold. $\lhd$

Now, let us prove Proposition 1 using the above lemma.

($\Rightarrow$) Assume that $\mathcal{G}_{\texttt{lgg}}$ is an lgg of $\mathcal{G}_1, \ldots, \mathcal{G}_n$ and let us show that it is also an lgg of some lgg of $\mathcal{G}_1, \ldots, \mathcal{G}_{n-1}$ and $\mathcal{G}_n$. By Definition 1, because $\mathcal{G}_{\texttt{lgg}}$ is an lgg of $\mathcal{G}_1, \ldots, \mathcal{G}_n$, we have: (*i*) $\mathcal{G}_{1\leq i\leq n} \models_{\mathcal{R}} \mathcal{G}_{\texttt{lgg}}$ holds and (*ii*) for any RDF graph $\mathcal{G}$ such that $\mathcal{G}_{1\leq i\leq n} \models_{\mathcal{R}} \mathcal{G}$ holds, $\mathcal{G}_{\texttt{lgg}} \models_{\mathcal{R}} \mathcal{G}$ holds.

Let $\mathcal{G}'_{\texttt{lgg}}$ be some lgg of $\mathcal{G}_1, \ldots, \mathcal{G}_{n-1}$. From (*i*) and the above lemma, (*) $\mathcal{G}'_{\texttt{lgg}} \models_{\mathcal{R}} \mathcal{G}_{\texttt{lgg}}$ and $\mathcal{G}_n \models_{\mathcal{R}} \mathcal{G}_{\texttt{lgg}}$ holds.

Moreover, for any RDF graph $\mathcal{G}$, if it were that $\mathcal{G}'_{\texttt{lgg}} \models_{\mathcal{R}} \mathcal{G}$ and $\mathcal{G}_n \models_{\mathcal{R}} \mathcal{G}$ and $\mathcal{G}_{\texttt{lgg}} \not\models_{\mathcal{R}} \mathcal{G}$, then $\mathcal{G}_{1\leq i\leq n} \models_{\mathcal{R}} \mathcal{G}$ would hold and contradict the fact that $\mathcal{G}_{\texttt{lgg}}$ is an lgg of $\mathcal{G}_1, \ldots, \mathcal{G}_n$. Therefore, (**) for any RDF graph $\mathcal{G}$, if $\mathcal{G}'_{\texttt{lgg}} \models_{\mathcal{R}} \mathcal{G}$ and $\mathcal{G}_n \models_{\mathcal{R}} \mathcal{G}$ holds, then $\mathcal{G}_{\texttt{lgg}} \models_{\mathcal{R}} \mathcal{G}$ holds.

From Definition 1, (*) and (**) we get that $\mathcal{G}_{\texttt{lgg}}$ is an lgg of an lgg of $\mathcal{G}_1, \ldots, \mathcal{G}_{n-1}$ and $\mathcal{G}_n$.

($\Leftarrow$) Assume that $\mathcal{G}_{\texttt{lgg}}$ is an lgg of an lgg $\mathcal{G}'$ of $\mathcal{G}_1, \ldots, \mathcal{G}_{n-1}$ and $\mathcal{G}_n$, and let us show that it is also an lgg of $\mathcal{G}_1, \ldots, \mathcal{G}_n$.

By Definition 1, (*i*) $\mathcal{G}' \models_{\mathcal{R}} \mathcal{G}_{\texttt{lgg}}$ and $\mathcal{G}_n \models_{\mathcal{R}} \mathcal{G}_{\texttt{lgg}}$ hold, hence $\mathcal{G}_{1\leq i\leq n-1} \models_{\mathcal{R}} \mathcal{G}_{\texttt{lgg}}$ and $\mathcal{G}_n \models_{\mathcal{R}} \mathcal{G}_{\texttt{lgg}}$ hold, i.e., (*') $\mathcal{G}_{1\leq i\leq n} \models_{\mathcal{R}} \mathcal{G}_{\texttt{lgg}}$ holds.

Moreover, (*ii*) for any RDF graph $\mathcal{G}$ such that $\mathcal{G}' \models_{\mathcal{R}} \mathcal{G}$ and $\mathcal{G}_n \models_{\mathcal{R}} \mathcal{G}$ hold, $\mathcal{G}_{\texttt{lgg}} \models_{\mathcal{R}} \mathcal{G}$ holds. By Definition 1, $\mathcal{G}_{1\leq i\leq n-1} \models_{\mathcal{R}} \mathcal{G}'$ holds, therefore we get: (**') for any RDF graph $\mathcal{G}$ such that $\mathcal{G}_{1\leq i\leq n} \models_{\mathcal{R}} \mathcal{G}$ holds, $\mathcal{G}_{\texttt{lgg}} \models_{\mathcal{R}} \mathcal{G}$ holds.

From Definition 1, (*') and (**') we get that $\mathcal{G}_{\texttt{lgg}}$ is an lgg of $\mathcal{G}_1, \ldots, \mathcal{G}_n$. $\square$

Based on the above result, without loss of generality, we focus in the next section on the following problem:

PROBLEM 1. *Given* two *RDF graphs* $\mathcal{G}_1, \mathcal{G}_2$ *and a set* $\mathcal{R}$ *of RDF entailment rules, we want to compute* some lgg *of* $\mathcal{G}_1$ *and* $\mathcal{G}_2$.

3.2 Computing an lgg of RDF graphs

We first devise the *cover graph* of two RDF graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ (to be defined shortly, Definition 2 below), which is central to our technique for computing an lgg of $\mathcal{G}_1$ and $\mathcal{G}_2$. We indeed show (Theorem 2) that this particular RDF graph corresponds to an lgg of $\mathcal{G}_1$ and $\mathcal{G}_2$ when considering their explicit triples *only*, i.e., ignoring RDF entailment rules. Then, we show the main result of this section (Theorem 3): an lgg of $\mathcal{G}_1$ and $\mathcal{G}_2$, for *any set* $\mathcal{R}$ of RDF entailment rules, is the cover graph of their saturations w.r.t. $\mathcal{R}$. We also provide the worst-case size of cover graph-based lggs, as well as the worst-case time to compute them.

DEFINITION 2 (COVER GRAPH). *The* cover graph $\mathcal{G}$ *of two RDF graph* $\mathcal{G}_1$ *and* $\mathcal{G}_2$ *is the RDF graph such that for every property* p *in both* $\mathcal{G}_1$ *and* $\mathcal{G}_2$*:*

$$(t_1, p, t_2) \in \mathcal{G}_1 \text{ and } (t_3, p, t_4) \in \mathcal{G}_2 \text{ iff } (t_5, p, t_6) \in \mathcal{G}$$

with $t_5 = t_1$ *if* $t_1 = t_3$ *and* $t_1 \in \mathcal{U} \cup \mathcal{L}$*, else* t_5 *is the blank node* $b_{t_1 t_3}$*, and, similarly* $t_6 = t_2$ *if* $t_2 = t_4$ *and* $t_2 \in \mathcal{U} \cup \mathcal{L}$*, else* t_6 *is the blank node* $b_{t_2 t_4}$.

The cover graph is more general than $\mathcal{G}_1$ and $\mathcal{G}_2$ as each of its triple (t_5, p, t_6) is a *least general anti-unifier* of a triple (t_1, p, t_2) from $\mathcal{G}_1$ and a triple (t_3, p, t_4) from $\mathcal{G}_2$. The notion of *least general anti-unifier* [39, 40, 43] is dual to the well-known notion of *most general unifier* [42, 43]. Observe that $\mathcal{G}$'s triples result from anti-unifications of $\mathcal{G}_1$ and $\mathcal{G}_2$ triples with *same* property URI. Indeed, anti-unifying triples of the form $(\texttt{s}_1, p, \texttt{o}_1)$ and $(\texttt{s}_2, p', \texttt{o}_2)$, with $p \neq p'$, would lead to a non-well-formed triples of the form $(\texttt{s}, b_{pp'}, \texttt{o})$ (recall that property values *must* be URIs in RDF graphs), where $b_{pp'}$ is the blank node required to generalize the distinct values p and p'.

Further, the cover graph is an lgg for the explicit triples in $\mathcal{G}_1$ and those in $\mathcal{G}_2$ since, intuitively, we capture their *common structures* by consistently naming, across all the anti-unifications begetting $\mathcal{G}$, the blank nodes used to generalize pairs of distinct subject values or of object values: each time the distinct values t from $\mathcal{G}_1$ and t' from $\mathcal{G}_2$ are generalized by a blank node while anti-unifying two triples, it is *always* by the same blank node $b_{t,t'}$ in $\mathcal{G}$. This way, we establish *joins* between $\mathcal{G}$ triples, which reflect the common join structure on t within $\mathcal{G}_1$ and on t' within $\mathcal{G}_2$. For instance in Figure 3, the *explicit* triples $(i_1, \tau, \text{ConfPaper})$, $(\text{ConfPaper}, \preceq_{\text{sc}}, \text{Publication})$, $(i_1, \text{title}, \text{"DiD"})$ in $\mathcal{G}_1$, and $(i_2, \tau, \text{JourPaper})$, $(\text{JourPaper}, \preceq_{\text{sc}}, \text{Publication})$, $(i_2, \text{title}, \text{"CwFOL"})$ in $\mathcal{G}_2$, lead to the triples $(b_{i_1 i_2}, \tau, b_{\text{CPJP}})$, $(b_{\text{CPJP}}, \preceq_{\text{sc}}, \text{Publication})$, $(b_{i_1 i_2}, \text{title}, b_{\text{DC}})$ in the cover graph of $\mathcal{G}_1$ and $\mathcal{G}_2$ shown in Figure 4 (left). The first above-mentioned $\mathcal{G}$ triple results from anti-unifying i_1 and i_2 into $b_{i_1 i_2}$, and, ConfPaper and JourPaper into b_{CPJP}. The second results from anti-unifying *again* ConfPaper and JourPaper into b_{CPJP}, and, Publication and Publication into Publication (as a constant is its own least general generalization). Finally, the third results from anti-unifying *again* i_1 and i_2 into $b_{i_1 i_2}$, and, "DiD" and "CwFOL" into b_{DC}. By reusing consistently the same blank node name $b_{i_1 i_2}$ for each anti-unification of the constants i_1 and i_2 (resp. b_{CPJP} for ConfPaper and JourPaper)), the cover graph triples join on $b_{i_1 i_2}$ (resp. b_{CPJP}) in order to reflect that, in $\mathcal{G}_1$ and in $\mathcal{G}_2$, there exists a particular case of publication (i_1 in $\mathcal{G}_1$ and i_2 in $\mathcal{G}_2$) with some title ("DiD" in $\mathcal{G}_1$ and "CwFOL" in $\mathcal{G}_2$).

The next theorem formalizes the above discussion by stating that the cover graph of two RDF graphs is an lgg of them, *just in case of an empty set of RDF entailment rules.*

THEOREM 2. *The* cover graph $\mathcal{G}$ *of the RDF graphs* $\mathcal{G}_1$ *and* $\mathcal{G}_2$ *is an* lgg *of them for the empty set* $\mathcal{R}$ *of RDF entailment rules (i.e.,* $\mathcal{R} = \emptyset$*).*

PROOF. The proof can be directly derived from that of the more general Theorem 3 (see below), noting that in the particular case of Theorem 2 $\mathcal{R} = \emptyset$ holds, hence $\mathcal{G}_1^\infty = \mathcal{G}_1$ and $\mathcal{G}_2^\infty = \mathcal{G}_2$ holds. $\square$

We provide below worst-case bounds for the time to compute a cover graph and for its size; these bounds are met when *all the triples of the two input graphs use the same property URI* (i.e., every pair of $\mathcal{G}_1$ and $\mathcal{G}_2$ triples begets a $\mathcal{G}$ triple).

PROPOSITION 2. *The cover graph of two RDF graphs* $\mathcal{G}_1$ *and* $\mathcal{G}_2$ *can be computed in* $O(|\mathcal{G}_1| \times |\mathcal{G}_2|)$*; its size is bounded by* $|\mathcal{G}_1| \times |\mathcal{G}_2|$.

The main theorem below generalizes Theorem 2 in order to take into account any set of entailment rules from the RDF standard. It states that it is sufficient to compute the cover of the saturations of the input RDF graphs, instead of the input RDF graphs themselves.

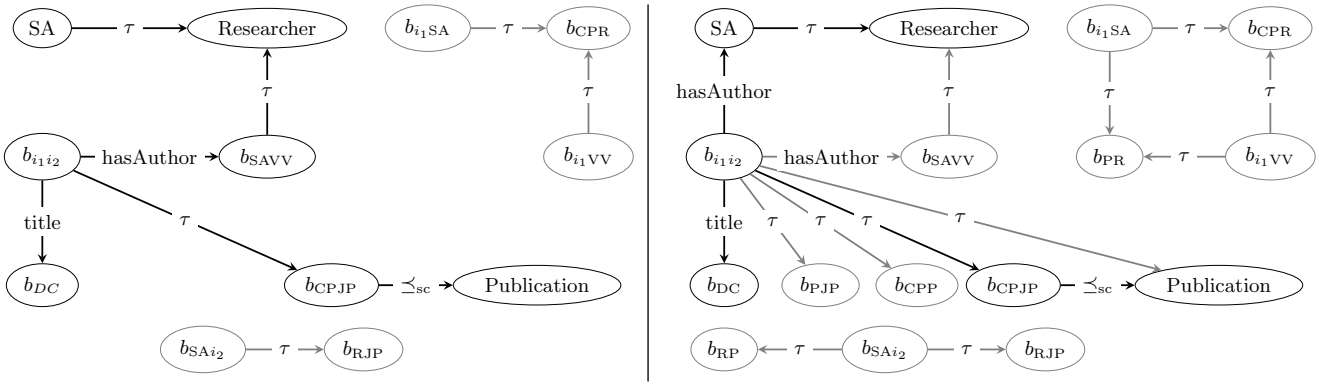

Figure 4: Cover graphs of $\mathcal{G}_1$ and $\mathcal{G}_2$ in Figure 3 (left) and of their saturations w.r.t. the entailment rules in Table 2 (right). Triples shown in grey are redundant w.r.t. those shown in black: they are part of the graph, while implicit to the triples shown in black.

THEOREM 3. *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two RDF graphs, and $\mathcal{R}$ a set of RDF entailment rules. The* cover graph $\mathcal{G}$ *of $\mathcal{G}_1^\infty$ and $\mathcal{G}_2^\infty$ is an* lgg *of $\mathcal{G}_1$ and $\mathcal{G}_2$.*

PROOF. We start by showing that $\mathcal{G}$ is a generalization of $\mathcal{G}_1$ and $\mathcal{G}_2$, i.e., $\mathcal{G}_1 \models_\mathcal{R} \mathcal{G}$ *and* that $\mathcal{G}_2 \models_\mathcal{R} \mathcal{G}$. Consider the RDF graph $\mathcal{G}'$ obtained from $\mathcal{G}$ by replacing every blank node $b_{v_1 v_2}$ by the value v_1. Clearly, $\mathcal{G}' \models_\mathcal{R} \mathcal{G}$ and, by construction of $\mathcal{G}$, $\mathcal{G}' = \mathcal{G}_1$, i.e., $\mathcal{G}_1 \models_\mathcal{R} \mathcal{G}$. Showing $\mathcal{G}_2 \models_\mathcal{R} \mathcal{G}$ is done in a similar way. Hence, $\mathcal{G}$ is a generalization of $\mathcal{G}_1$ and $\mathcal{G}_2$.

Let us show now that $\mathcal{G}$ is an lgg of $\mathcal{G}_1$ and $\mathcal{G}_2$. Let $\mathcal{G}_{\text{lgg}}$ be *any* lgg of $\mathcal{G}_1$ and $\mathcal{G}_2$. To prove our claim, we need to show that $\mathcal{G} \models_\mathcal{R} \mathcal{G}_{\text{lgg}}$ holds. Since $\mathcal{G}_{\text{lgg}}$ is an lgg of $\mathcal{G}_1$ and $\mathcal{G}_2$, there exist two homomorphisms ϕ_1 and ϕ_2 from the blank nodes in $\mathcal{G}_{\text{lgg}}$ to the values in $\mathcal{G}_1^\infty$ and in $\mathcal{G}_2^\infty$ respectively, such that $[\mathcal{G}_{\text{lgg}}]_{\phi_1} \subseteq \mathcal{G}_1^\infty$ and $[\mathcal{G}_{\text{lgg}}]_{\phi_2} \subseteq \mathcal{G}_2^\infty$. Consider the graph $\mathcal{G}'_{\text{lgg}}$ obtained from $\mathcal{G}_{\text{lgg}}$ by replacing every blank node b by $b_{v_1 v_2}$ where $v_1 = \phi_1(b)$ and $v_2 = \phi_2(b)$. Clearly, $\mathcal{G}_{\text{lgg}} \equiv_\mathcal{R} \mathcal{G}'_{\text{lgg}}$ holds. Let us show that $\mathcal{G} \models_\mathcal{R} \mathcal{G}'_{\text{lgg}}$ holds, i.e., there exists a homomorphism ϕ from the blank nodes in $\mathcal{G}'_{\text{lgg}}$ to the terms in $\mathcal{G}^\infty$ such that $[\mathcal{G}'_{\text{lgg}}]_\phi \subseteq \mathcal{G}^\infty$. By construction of $\mathcal{G}$, it is easy to see that the above holds when ϕ maps $b_{v_1 v_2}$ either to v_1 if $v_1 = v_2$ and $v_1 \in \mathcal{U} \cup \mathcal{L}$, or to itself otherwise. It therefore follows that $\mathcal{G} \models_\mathcal{R} \mathcal{G}'_{\text{lgg}}$, hence $\mathcal{G} \models_\mathcal{R} \mathcal{G}_{\text{lgg}}$. □

As an immediate consequence of the above results, we get the following worst-case bounds for the time to compute a cover graph-based lgg of two RDF graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, and for its size. Here, we assume given the saturation $\mathcal{G}_1^\infty$ and $\mathcal{G}_2^\infty$, as the times to compute them and their sizes depend on the RDF entailment rules at hand.

COROLLARY 1. *An* lgg *of two RDF graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ can be computed in $O(|\mathcal{G}_1^\infty| \times |\mathcal{G}_2^\infty|)$ and its size is bounded by $|\mathcal{G}_1^\infty| \times |\mathcal{G}_2^\infty|$.*

Remark that computing naively the cover graph-based lgg of n RDF graphs of size M based on Proposition 1 may lead to an lgg of size M^n, in the unlikely worst-case where all the triples of all the RDF graphs use the same property URI. However, removing the redundant triples from the intermediate and final cover graph-based lggs limits their size to at most M.

Figure 4 (right) displays an lgg of the RDF graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ in Figure 3 w.r.t. the entailment rules shown in Table 2. In contrast to Figure 4 (left), which shows an lgg of the same RDF graphs when RDF entailment rules are ignored, we further learn that Serge Abiteboul is an author of some particular publication (i_1 in $\mathcal{G}_1$ and i_2 in $\mathcal{G}_2$). Moreover, removing the redundant triples (the grey ones) yields precisely the lgg $\mathcal{G}_{\text{lgg}}$ of $\mathcal{G}_1$ and $\mathcal{G}_2$ shown in Figure 3.

4. FINDING COMMONALITIES BETWEEN QUERIES

We consider now the problem of finding the largest set of commonalities between queries. Adapting the results for RDF graphs from the preceding section to BGPQs is rather direct (recall that a query body is a BGP, i.e., an RDF graph in which variables can be used in subject, property and object positions of triples), but of limited interest as the next example shows. Consider the two BGPQs q_1 and q_2 in Figure 6. The first asks for the conference papers having some contact author, while the second asks for the journal papers having some author. Clearly, a least general generalization of q_1 and q_2 is the *very* general BGPQ q_{lgg} shown in Figure 6 that asks for the resources having some type. Observe that q_{lgg} is an lgg for *any* subset of the RDF entailment rules shown in Table 2, as there is *no* ontological constraint in the queries. However, by considering the extra ontological constraints displayed in Figure 7 that hold in the scientific publication domain, i.e., the context in which the queries are asked, a more pregnant lgg would be a BGPQ asking for *publications having some researcher as author*, since (*i*) having a contact authors is having an author, (*ii*) only publications have authors, (*iii*) only researchers are authors, and (*iv*) conference (resp. journal) papers are publications.

We therefore devise in Section 4.1 a notion of lgg of BGPQs in the presence of ontological constraints, which as exemplified above helps finding more interesting commonalities by taking into account the background knowledge of an application domain. Then, we provide a technique for computing such an lgg of BGPQs in Section 4.2.

4.1 Defining the lgg of BGPQs

Recall that the notion of lgg of n descriptions $d_1, \ldots, d_n$ is a most specific description d generalizing $d_1, \ldots, d_n$ for some generalization/specialization relation. Here, we adapt this notion by using BGPQs as descriptions and revisiting en-

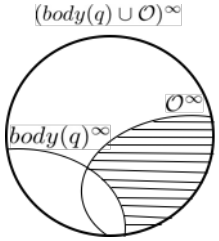

Figure 5: Characterization of the body of a saturated BGPQ q w.r.t. a set $\mathcal{O}$ of RDFS constraints

tailment between BGPQs (Section 2) in order to endow this RDF relation between queries with background knowledge.

We first devise in Section 4.1.1 *entailment between BGPQs w.r.t. ontological constraints* as a generalization of the standard entailment between BGPQs. Then, we define the `lgg` *of BGPQs w.r.t. ontological constraints* in Section 4.1.2.

4.1.1 Entailment between BGPQs w.r.t. constraints

We start by generalizing the saturation of a query w.r.t. extra ontological constraints. The saturated query comprises all the triples in the saturation of its body augmented with the constraints, except those triples that only follow from the ontological constraints, i.e., which are not related to what the query is asking for.

Definition 3 (Saturation w.r.t. constraints). *Let $\mathcal{R}$ be a set of RDF entailment rules, $\mathcal{O}$ a set of RDFS statements, and q a BGPQ. The* saturation *of q w.r.t. $\mathcal{O}$, denoted $q^\infty_\mathcal{O}$, is a BGPQ with the same answer variables as q and whose body, denoted $body(q^\infty_\mathcal{O})$, is the maximal subset of $(\mathcal{O} \cup body(q))^\infty$ such that for any of its subset $\mathcal{S}$: if $\mathcal{O} \models_\mathcal{R} \mathcal{S}$ holds then $body(q) \models_\mathcal{R} \mathcal{S}$ holds.*

The above definition precisely characterizes $body(q^\infty_\mathcal{O})$ as the *non-hatched* subset of $(\mathcal{O} \cup body(q))^\infty$ shown in Figure 5. Also, remark that this definition coincides with the standard one of saturation (Section 2) when the set $\mathcal{O}$ of ontological constraints is empty.

Figure 7 shows a set $\mathcal{O}$ of ontological constraints, as well as the saturations of the BGPQs q_1 and q_2 in Figure 6 w.r.t. $\mathcal{O}$, named $q_1{}^\infty_\mathcal{O}$ and $q_2{}^\infty_\mathcal{O}$ respectively.

The theorem below states the fundamental properties for a query and its saturation w.r.t. ontological constraints: they are *equivalent* for the central RDF reasoning tasks of query entailment and query answering.

Theorem 4. *Let $\mathcal{R}$ be a set of RDF entailment rules, $\mathcal{O}$ a set of RDFS statements, and q a BGPQ whose saturation w.r.t. $\mathcal{O}$ is $q^\infty_\mathcal{O}$. For any RDF graph $\mathcal{G}$ whose set of RDFS statements is $\mathcal{O}$,*

1. *$\mathcal{G} \models_\mathcal{R} q$ holds iff $\mathcal{G} \models_\mathcal{R} q^\infty_\mathcal{O}$ holds,*
2. *$q(\mathcal{G}) = q^\infty_\mathcal{O}(\mathcal{G})$ holds.*

Proof. Let us first show item 1). Observe that $\mathcal{G} \models_\mathcal{R} q$ *iff* $\mathcal{G} \models_\mathcal{R} q^\infty_\mathcal{O}$ holds iff $\mathcal{G}^\infty \models q$ *iff* $\mathcal{G}^\infty \models q^\infty_\mathcal{O}$ holds (recall Section 2). If $\mathcal{G}^\infty \models^\phi q^\infty_\mathcal{O}$ holds for some homomorphism ϕ, then $\mathcal{G}^\infty \models^\phi q$ holds also since $body(q) \subseteq body(q^\infty_\mathcal{O})$. Now, if $\mathcal{G}^\infty \models^\phi q$ holds for some homomorphism ϕ, consider the subset $[body(q)]_\phi$ of $\mathcal{G}^\infty$. Since $\mathcal{O} \subseteq \mathcal{G}$, and by definition of the saturation of q w.r.t. $\mathcal{O}$, ϕ can obviously be extended to a homomorphism ϕ' such that $[body(q^\infty_\mathcal{O})]_{\phi'} \subseteq \mathcal{G}^\infty$, which maps $body(q^\infty_\mathcal{O})$ to the maximal subset of $(\mathcal{O} \cup [body(q)]_\phi)^\infty \subseteq \mathcal{G}^\infty$, such that for any of its subset $\mathcal{G}'$: if $\mathcal{O} \models_\mathcal{R} \mathcal{G}'$ then $[body(q)]_\phi \models_\mathcal{R} \mathcal{G}'$.

Item 2) directly follows from: (i) the fact that, above, item 1) holds for any homomorphism ϕ and (ii) q and $q^\infty_\mathcal{O}$ have, by definition, the same answer variables. □

Based on the above result, we are now ready to extend the entailment relation between queries, i.e., the standard RDF generalization/specialization relation between queries, to that of *entailment between queries w.r.t. ontological constraints*.

Definition 4 (Entailment w.r.t. constraints). *Given a set $\mathcal{R}$ of RDF entailment rules, a set $\mathcal{O}$ of RDFS statements, and two BGPQs q and q' with the same arity, q* entails *q' w.r.t. $\mathcal{O}$, denoted $q \models_{\mathcal{R},\mathcal{O}} q'$, iff $q^\infty_\mathcal{O} \models q'$ holds.*

The above definition coincides with the standard one of entailment between BGPQs (Section 2) when the set $\mathcal{O}$ of ontological constraints is empty.

The BGPQ $q(x) \leftarrow (x, \tau, \text{Publication}), (x, \text{hasAuthor}, y)$ is *not* entailed by the queries q_1 and q_2 shown in Figure 6, while it is entailed by these queries w.r.t. the set $\mathcal{O}$ of ontological constraints displayed in Figure 7: $q_1{}^\infty_\mathcal{O} \models^{\phi_1} q$ (resp. $q_2{}^\infty_\mathcal{O} \models^{\phi_2} q$) holds for $\phi_1(x) = x_1$ and $\phi_1(y) = y_1$ (resp. $\phi_2(x) = x_2$ and $\phi_2(y) = y_2$).

The next theorem establishes the fundamental properties for a query entailed by another w.r.t. ontological constraints: the former *generalizes* the latter for the central RDF reasoning tasks of query entailment and query answering.

Theorem 5. *Let $\mathcal{R}$ be a set of RDF entailment rules, $\mathcal{O}$ a set of RDFS statements, and two BGPQs q and q' such that $q \models_{\mathcal{R},\mathcal{O}} q'$. For any RDF graph $\mathcal{G}$ whose set of RDFS statements is $\mathcal{O}$,*

1. *if $\mathcal{G} \models_\mathcal{R} q$ holds then $\mathcal{G} \models_\mathcal{R} q'$ holds,*
2. *$q(\mathcal{G}) \subseteq q'(\mathcal{G})$ holds.*

Proof. Let us first show item 1). By Theorem 4, item 1), $\mathcal{G} \models_\mathcal{R} q$ holds iff $\mathcal{G} \models_\mathcal{R} q^\infty_\mathcal{O}$ holds. Consider some ϕ such that $\mathcal{G} \models^\phi_\mathcal{R} q^\infty_\mathcal{O}$ holds. Also, by Definition 4, consider some ϕ' such that $q^\infty_\mathcal{O} \models^{\phi'} q'$. By composing ϕ and ϕ' into ψ, we get $\mathcal{G} \models^\psi_\mathcal{R} q'$, i.e., $\mathcal{G} \models_\mathcal{R} q'$.

Item 2) directly follows from: (i) the fact that, above, item 1) holds for any pair of homomorphisms ϕ, ϕ' such that $\mathcal{G} \models^\phi_\mathcal{R} q^\infty_\mathcal{O}$ and $q^\infty_\mathcal{O} \models^{\phi'} q'$ hold, and (ii) q and $q^\infty_\mathcal{O}$ have, by definition, the same answer variables. □

Finally, it is worth noting that our notion of entailment between BGPQs w.r.t. ontological constraints is the RDF counterpart to that of *query containment under deductive constraints* in a database setting [16].

4.1.2 `lgg` of BGPQs w.r.t. constraints

With the above new RDF generalization/specialization relation between queries endowed with background knowledge, we are now ready to define the `lgg` of queries w.r.t. ontological constraints.

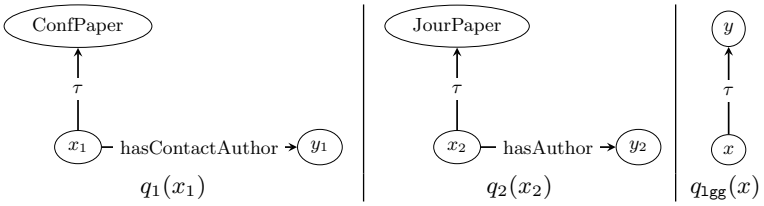

Figure 6: Sample BGPQs q_1, q_2 and their minimal lgg $q_{\texttt{lgg}}$.

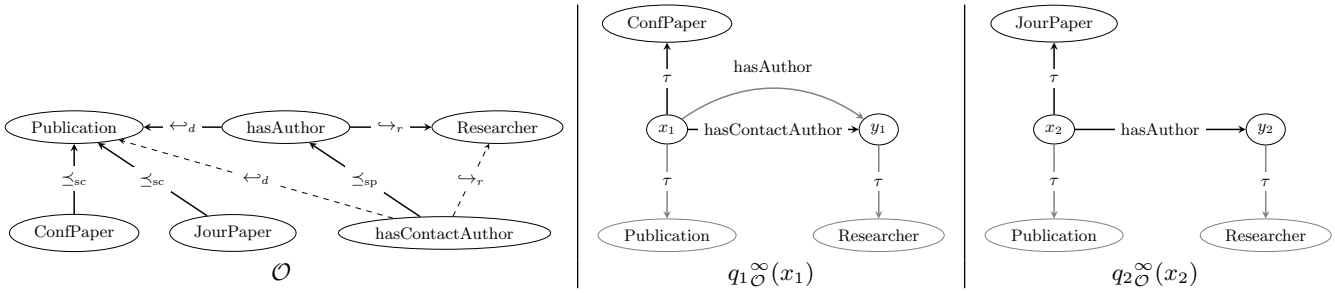

Figure 7: Sample set of ontological constraints $\mathcal{O}$; saturations of the BGPQs q_1 and q_2 in Figure 6 w.r.t. $\mathcal{O}$. Triples shown in grey are redundant w.r.t. those shown in black.

Definition 5 (lgg of BGPQs). *Let $q_1, \ldots, q_n$ be BGPQs with the same arity, $\mathcal{R}$ a set of RDF entailment rules and $\mathcal{O}$ a set of RDFS statements.*

- *A* generalization *of $q_1, \ldots, q_n$ w.r.t. $\mathcal{O}$ is a BGPQ q_g such that $q_i \models_{\mathcal{R},\mathcal{O}} q_g$ holds for $1 \leq i \leq n$.*
- *A* least general generalization *of $q_1, \ldots, q_n$ w.r.t. $\mathcal{O}$ is a generalization $q_{\texttt{lgg}}$ of $q_1, \ldots, q_n$ w.r.t. $\mathcal{O}$ such that for any other generalization q_g of $q_1, \ldots, q_n$ w.r.t. $\mathcal{O}$, $q_{\texttt{lgg}} \models_{\mathcal{R},\mathcal{O}} q_g$ holds.*

When $\mathcal{O}$ is empty, the above definition is the direct adaptation of that of lgg of RDF graphs to the case of BGPQs, using standard entailment between BGPQs.

Importantly, in our RDF and SPARQL setting, the following holds:

Theorem 6. *An lgg of BGPQs w.r.t. RDFS statements may not exist for some set of RDF entailment rules; when it exists, it is unique up to entailment ($\models_{\mathcal{R},\mathcal{O}}$).*

Proof. It is easy to show that an lgg of BGPQs may not exist. For instance, consider the BGPQs $q_1(x_1) \leftarrow (x_1, \text{hasAuthor}, y_1)$ asking for the resources having some author, and $q_2(x_2) \leftarrow (y_2, \text{hasAuthor}, x_2)$ asking for the authors of some resource. Clearly, when the set $\mathcal{R}$ of entailment rules is empty or comprises the rules in Table 2, *no* BGPQ can generalize q_1 and q_2, hence there is *no* lgg of them. By contrast, if we use the complete set of RDF entailment rules, an lgg of q_1 and q_2 is $q_{\texttt{lgg}}(x) \leftarrow (x, \tau, \text{rdf:Resource})$, since every RDF value is an instance of the built-in class rdf:Resource.

When an lgg of BGPQs w.r.t. RDFS constraints exists, it is unique up to entailment, i.e., is semantically unique, because $q_{\texttt{lgg}} \models_{\mathcal{R},\mathcal{O}} q_g$ holds for any q_g in Definition 5. If it were that queries have *multiples* lggs *incomparable* w.r.t. entailment, say the BGPQs $\texttt{lgg}_1(\bar{x}), \ldots, \texttt{lgg}_m(\bar{x})$, the BGPQ defined as $q_{\texttt{lgg}}(\bar{x}) \leftarrow \mathit{body}(\texttt{lgg}_1) \cup \cdots \cup \mathit{body}(\texttt{lgg}_m)$ would be a *single strictly more specific* lgg, a contradiction. □

Though unique up to entailment, there exist many syntactic variants (an infinity actually) of an lgg due to *redundant* triples, i.e., triples entailed by others within the lgg. For example, think of an lgg $q_{\texttt{lgg}}(x) \leftarrow (x, \tau, A), (x, \tau, B), (x, y, z)$ w.r.t. the set of constraints $\mathcal{O} = \{(A, \prec_{\text{sc}}, B), (B, \prec_{\text{sc}}, A)\}$, which asks for resources of types A and B that are somehow related to some resource, and it is known that A and B are equivalent classes. Clearly, different equivalent and minimal variants (w.r.t. the number of triples) of this lgg are $q_{\texttt{lgg}}(x) \leftarrow (x, \tau, A)$ and $q_{\texttt{lgg}}(x) \leftarrow (x, \tau, B)$, since (x, y, z) is entailed by each of the two other triples, and (x, τ, B) is entailed by (x, τ, A) w.r.t. $\mathcal{O}$, and vice versa, because A and B are equivalent. Importantly, redundancy of triples is not specific to lggs of BGPQs w.r.t. RDFS constraints, since obviously any BGPQ may feature redundancy. The detection and elimination of such redundancy have been studied in the literature, hence we focus in this work on learning *some* lgg of BGPQs w.r.t. RDFS constraints; learning as minimal as possible lggs is a perspective of this work discussed in Section 8.

The proposition below is the counterpart of Proposition 1 for RDF graphs. It shows that an lgg of n BGPQs, with $n \geq 3$, can be defined (hence computed) as a sequence of $n-1$ lggs of *two* BGPQs. That is, assuming that $\ell_{k \geq 2}$ is an operator computing an lgg of k input BGPQs, the next proposition establishes that:

$$\begin{array}{ll} \ell_3(q_1, q_2, q_3) & \equiv_{\mathcal{R},\mathcal{O}} \ \ell_2(\ell_2(q_1, q_2), q_3) \\ \ldots & \ldots \\ \ell_n(q_1, \ldots, q_n) & \equiv_{\mathcal{R},\mathcal{O}} \ \ell_2(\ell_{n-1}(q_1, \ldots, q_{n-1}), q_n) \\ & \equiv_{\mathcal{R},\mathcal{O}} \ \ell_2(\ell_2(\cdots \ell_2(\ell_2(q_1, q_2), q_3) \cdots, q_{n-1}), q_n) \end{array}$$

Proposition 3. *Let $q_1, \ldots, q_{n \geq 3}$ be n BGPQs, $\mathcal{O}$ a set of RDFS statements and $\mathcal{R}$ a set of RDF entailment rules.*

$q_{1\text{gg}}$ is an lgg *of $q_1, \ldots, q_n$ w.r.t. $\mathcal{O}$ iff $q_{1\text{gg}}$ is an* lgg *w.r.t. $\mathcal{O}$ of an* lgg *of $q_1, \ldots, q_{n-1}$ and q_n w.r.t. $\mathcal{O}$.*

PROOF. The proof relies on the next lemma.

LEMMA 2. *Let $q_1, \ldots, q_n$ be BGPQs, $\mathcal{O}$ a set of RDFS statements and $\mathcal{R}$ a set of RDF entailment rules. If $q^1_{1\text{gg}}$ is an* lgg *of $q_1, \ldots, q_{k<n}$ w.r.t. $\mathcal{O}$ and $q^2_{1\text{gg}}$ is an* lgg *of $q_1, \ldots, q_n$ w.r.t. $\mathcal{O}$, then $q^1_{1\text{gg}} \models_{\mathcal{R},\mathcal{O}} q^2_{1\text{gg}}$ holds.*

Let us show that the above lemma holds.

Suppose that $q^1_{1\text{gg}}$ is an lgg of $q_1, \ldots, q_{k<n}$ w.r.t. $\mathcal{O}$, i.e., by Definition 5: (i) $q_{1\leq i\leq k} \models_{\mathcal{R},\mathcal{O}} q^1_{1\text{gg}}$ holds and (ii) for any BGPQ q such that $q_{1\leq i\leq k} \models_{\mathcal{R},\mathcal{O}} q$ holds, $q^1_{1\text{gg}} \models_{\mathcal{R},\mathcal{O}} q$ holds.

Suppose also that $q^2_{1\text{gg}}$ is an lgg of $q_1, \ldots, q_n$ w.r.t. $\mathcal{O}$, i.e., by Definition 5: (i) $q_{1\leq i\leq n} \models_{\mathcal{R},\mathcal{O}} q^2_{1\text{gg}}$ holds and (ii) for any BGPQ q' such that $q_{1\leq i\leq n} \models_{\mathcal{R},\mathcal{O}} q'$ holds, $q^2_{1\text{gg}} \models_{\mathcal{R},\mathcal{O}} q'$ holds.

Clearly, $q^2_{1\text{gg}}$ is a possible value for q above, because $q_{1\leq i\leq n} \models_{\mathcal{R},\mathcal{O}} q^2_{1\text{gg}}$ implies $q_{1\leq i\leq k<n} \models_{\mathcal{R},\mathcal{O}} q^2_{1\text{gg}}$, hence $q^1_{1\text{gg}} \models_{\mathcal{R},\mathcal{O}} q^2_{1\text{gg}}$ must hold. $\lhd$

Now, let us prove Proposition 3 using the above lemma.

$(\Rightarrow)$ Assume that $q_{1\text{gg}}$ is an lgg of $q_1, \ldots, q_n$ w.r.t. $\mathcal{O}$ and let us show that it is also an lgg w.r.t. $\mathcal{O}$ of some lgg of $q_1, \ldots, q_{n-1}$ and q_n w.r.t. $\mathcal{O}$. By Definition 5, because $q_{1\text{gg}}$ is an lgg of $q_1, \ldots, q_n$ w.r.t. $\mathcal{O}$, we have: (i) $q_{1\leq i\leq n} \models_{\mathcal{R},\mathcal{O}} q_{1\text{gg}}$ holds and (ii) for any BGPQ q such that $q_{1\leq i\leq n} \models_{\mathcal{R},\mathcal{O}} q$ holds, $q_{1\text{gg}} \models_{\mathcal{R},\mathcal{O}} q$ holds.

Let $q'_{1\text{gg}}$ be an lgg of $q_1, \ldots, q_{n-1}$ w.r.t. $\mathcal{O}$. From (i) and the above lemma, (*) $q'_{1\text{gg}} \models_{\mathcal{R},\mathcal{O}} q_{1\text{gg}}$ and $q_n \models_{\mathcal{R},\mathcal{O}} q_{1\text{gg}}$ hold. Moreover, for any BGPQ q, if it were that $q'_{1\text{gg}} \models_{\mathcal{R},\mathcal{O}} q$ and $q_n \models_{\mathcal{R},\mathcal{O}} q$ and $q_{1\text{gg}} \not\models_{\mathcal{R},\mathcal{O}} q$, then $q_{1\leq i\leq n} \models_{\mathcal{R},\mathcal{O}} q$ would hold and contradict the fact that $q_{1\text{gg}}$ is an lgg of $q_1, \ldots, q_n$ w.r.t. $\mathcal{O}$. Therefore, (**) for any BGPQ q, if $q'_{1\text{gg}} \models_{\mathcal{R},\mathcal{O}} q$ and $q_n \models_{\mathcal{R},\mathcal{O}} q$ holds, then $q_{1\text{gg}} \models_{\mathcal{R},\mathcal{O}} q$ holds.

From Definition 5, (*) and (**) we get that $q_{1\text{gg}}$ is an lgg w.r.t. $\mathcal{O}$ of an lgg of $q_1, \ldots, q_{n-1}$ and q_n w.r.t. $\mathcal{O}$.

$(\Leftarrow)$ Assume that $q_{1\text{gg}}$ is an lgg w.r.t. $\mathcal{O}$ of an lgg q' of $q_1, \ldots, q_{n-1}$ and q_n w.r.t. $\mathcal{O}$, and let us show that it is also an lgg of $q_1, \ldots, q_n$ w.r.t. $\mathcal{O}$.

By Definition 5, (i) $q' \models_{\mathcal{R},\mathcal{O}} q_{1\text{gg}}$ and $q_n \models_{\mathcal{R},\mathcal{O}} q_{1\text{gg}}$ hold, hence $q_{1\leq i\leq n-1} \models_{\mathcal{R},\mathcal{O}} q_{1\text{gg}}$ and $q_n \models_{\mathcal{R},\mathcal{O}} q_{1\text{gg}}$ hold, i.e., (*') $q_{1\leq i\leq n} \models_{\mathcal{R},\mathcal{O}} q_{1\text{gg}}$ holds.

Moreover, (ii) for any BGPQ q such that $q' \models_{\mathcal{R},\mathcal{O}} q$ and $q_n \models_{\mathcal{R},\mathcal{O}} q$ hold, $q_{1\text{gg}} \models_{\mathcal{R},\mathcal{O}} q$ holds. By Definition 5, $q_{1\leq i\leq n-1} \models_{\mathcal{R},\mathcal{O}} q'$ holds, therefore we get: (**') for any BGPQ q such that $q_{1\leq i\leq n} \models_{\mathcal{R},\mathcal{O}} q$ holds, $q_{1\text{gg}} \models_{\mathcal{R},\mathcal{O}} q$ holds.

From Definition 5, (*') and (**') we get that $q_{1\text{gg}}$ is an lgg of $q_1, \ldots, q_n$ w.r.t. $\mathcal{O}$. $\square$

Based on the above result, without loss of generality, we focus in the next section on the following problem:

PROBLEM 2. *Given* two *BGPQs q_1, q_2 with same arity, a set $\mathcal{O}$ of RDFS statements, and a set $\mathcal{R}$ of RDF entailment rules, we want to compute* some lgg *of q_1 and q_2 w.r.t. $\mathcal{O}$.*

4.2 Computing an lgg of BGPQs

We first devise the notion of *cover query* of two BGPQs q_1 and q_2 (to be defined shortly, Definition 6 below) and we show (Theorem 7) that it corresponds to an lgg of q_1 and q_2 just in case extra ontological constraints and RDF entailment are ignored. Then, we show (Theorem 8) that their enhanced lgg as defined in Definition 5, i.e., which does consider extra ontological constraints and RDF entailment, can be obtained as the cover query of the saturations of q_1 and of q_2 with the ontological constraints and RDF entailment rules at hand (Definition 3). We also provide the size of these cover query-based lggs, as well as the time to compute them.

DEFINITION 6 (COVER QUERY). *Let q_1, q_2 be two BGPQs with the same arity n.*
If there exists the BGPQ q such that

- *$(t_1, t_2, t_3) \in body(q_1)$ and $(t_4, t_5, t_6) \in body(q_2)$ iff $(t_7, t_8, t_9) \in body(q)$ with, for $1 \leq i \leq 3$, $t_{i+6} = t_i$ if $t_i = t_{i+3}$ and $t_i \in \mathcal{U} \cup \mathcal{L}$, otherwise t_{i+6} is the variable $v_{t_i t_{i+3}}$*
- *$head(q) = q(v_{x^1_1 x^1_2}, \ldots, v_{x^n_1 x^n_2})$ iff $head(q_1) = q(x^1_1, \ldots, x^n_1)$ and $head(q_2) = q(x^1_2, \ldots, x^n_2)$*

then q is the cover query *of q_1 and q_2.*

The cover query body is defined (first item above) as a *slight generalization of the cover graph of two RDF graphs* (Definition 1 and Theorem 2), so that it corresponds to an lgg of the bodies of q_1 and q_2 when both extra ontological constraints and RDF entailment are ignored. Recall that the bodies of the BGPQs are BGPs, which generalize RDF graphs by allowing variables in triples (Section 2). In particular, unknown values are allowed in the property position of BGP triples (using variables), whereas this was not possible in RDF graph triples in which the property values must be URIs. To generalize the results of Theorem 2 from RDF graphs to BGPs, (i) *variables* are used instead of blank nodes to generalize distinct values in the least general anti-unifications of triples begetting the cover query body (as they have the same semantics within a BGP, but only variables can be used at subject, object *and property* positions in triples), (ii) the cover query body comprises the least general anti-unifications of *every* q_1 triple with *every* q_2 triple instead of just those pairs of triples with same property URI (as now distinct property values can be generalized using variables), and (iii) again, across these anti-unifications characterizing the body of q, we consistently name the variables generalizing same pair of the distinct q_1 and q_2 values (so as to capture the common structure between the bodies of q_1 and q_2).

The cover query head is simply defined (second item in Definition 6) as the least general anti-unification of the q_1's head and q_2's head, in order to model the least general set of result tuples generalizing those of q_1 and of q_2.

Figure 8 (left) shows the cover query q^c of the BGPQs q_1 and q_2 displayed in Figure 6. The head $q^c(v_{x_1x_2})$ of q^c results from anti-unifying $q_1(x_2)$ and $q_2(x_2)$; its body consists of the triple $(v_{x_1x_2}, \tau, v_{\text{CPJP}})$, which results from anti-unifying $(x_1, \tau, \text{ConfPaper})$ in q_1 and $(x_2, \tau, \text{JourPaper})$ in q_2, of the triple $(v_{x_1x_2}, v_{\text{hCA}\tau}, v_{y_1\text{JP}})$, which results from anti-unifying $(x_1, \text{hasContactAuthor}, y_1)$ in q_1 and $(x_2, \tau, \text{JourPaper})$ in q_2 etc.

Importantly, a cover query of two BGPQs may not exist (If ... then ... in Definition 6). For instance, recall the BGPQs q_1 and q_2 previously introduced in subsection 4.1.2, with which we pointed out that an lgg may no exists. Their cover query does not exists either, since Definition 6 leads to $q(v_{x_1x_2}) \leftarrow (v_{x_1y_2}, \text{hasAuthor}, v_{y_1x_2})$ which is *not* a BGPQ

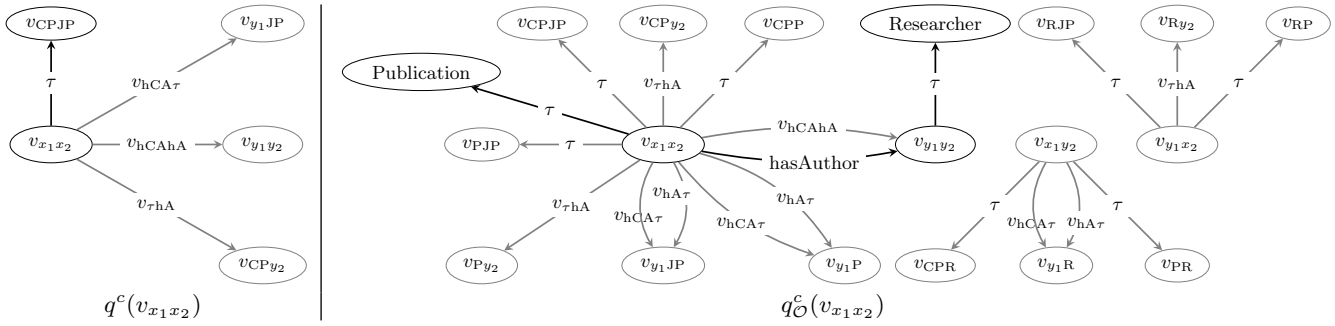

Figure 8: Cover queries of the BGPQs q_1 and q_2 in Figure 6 (left), and of their saturations w.r.t. $\mathcal{O}$ in Figure 7 (right). Triples shown in grey are redundant w.r.t. those shown in black.

($v_{x_1x_2}$ does not appear in q's body).

The next theorem formalizes the above discussion:

THEOREM 7. *Given two BGPQs q_1, q_2 with the same arity and empty sets $\mathcal{R}$ of RDF entailment rules and $\mathcal{O}$ of RDFS statements:*

1. *the cover query of q_1 and q_2 exists iff an* `lgg` *of q_1 and q_2 exists;*
2. *the cover query of q_1 and q_2 is an* `lgg` *of q_1 and q_2.*

PROOF. The proof can be directly derived from that of the more general Theorem 8 (see below), noting that in the particular case of Theorem 7 $\mathcal{R} = \emptyset$ and $\mathcal{O} = \emptyset$ hold, hence $q_1{}^{\infty}_{\mathcal{O}} = q_1$ and $q_2{}^{\infty}_{\mathcal{O}} = q_2$ hold. □

In Figure 8 (left), the cover query q^c of the BGPQs q_1 and q_2 displayed in Figure 6 is therefore an `lgg` of them when both extra ontological constraints and RDF entailment are ignored (Theorem 7). Remark that q^c is equivalent to the minimal `lgg` of q_1 and q_2 shown Figure 6: $q_{\texttt{lgg}}$.

We provide below the worst-case time to compute a cover query, and its size.

PROPOSITION 4. *The cover query of two BGPQs q_1 and q_2 can be computed in $O(|body(q_1)| \times |body(q_2)|)$; its size is $|body(q_1)| \times |body(q_2)|$.*

The next theorem generalizes the preceding one in order to use the notion of cover query to compute an `lgg` of two queries *w.r.t. extra ontological constraints* and *any* set of RDF entailment rules.

THEOREM 8. *Given a set $\mathcal{R}$ of RDF entailment rules, a set $\mathcal{O}$ of RDFS statements and two BGPQs q_1, q_2 with the same arity,*

1. *the cover query q of $q_1{}^{\infty}_{\mathcal{O}}$ and $q_2{}^{\infty}_{\mathcal{O}}$ exists iff an* `lgg` *of q_1 and q_2 w.r.t. $\mathcal{O}$ exists;*
2. *the cover query q of $q_1{}^{\infty}_{\mathcal{O}}$ and $q_2{}^{\infty}_{\mathcal{O}}$ is an* `lgg` *of q_1 and q_2 w.r.t. $\mathcal{O}$.*

PROOF. We start by showing that the cover query q, when it exists, is a generalization w.r.t. $\mathcal{O}$ of q_1 and q_2, i.e., $q_1 \models_{\mathcal{R},\mathcal{O}} q$ *and* that $q_2 \models_{\mathcal{R},\mathcal{O}} q$. Clearly, once proved, this entails de fact that if q exists, then an `lgg` of q_1 and q_2 w.r.t. $\mathcal{O}$ exists.

Consider the BGPQ q' obtained from q by replacing every variable $v_{t_1t_2}$ by the value t_1. Clearly, $q' \models_{\mathcal{R},\mathcal{O}} q$ and, by construction of q, $q' = q_1$, i.e., $q_1 \models_{\mathcal{R},\mathcal{O}} q$. Showing that $q_2 \models_{\mathcal{R},\mathcal{O}} q$ holds is done in a similar way.

Therefore, the cover query q is a generalization of q_1 and q_2 w.r.t. $\mathcal{O}$, thus an `lgg` of q_1 and q_2 w.r.t. $\mathcal{O}$ also exists.

Now, let us show that the cover query q, when it exists, is an `lgg` of q_1 and q_2 w.r.t. $\mathcal{O}$. Let $q_{\texttt{lgg}}$ be *any* `lgg` of q_1 and q_2 w.r.t. $\mathcal{O}$. Without loss of generality, assume that BGPQs do not contain blank nodes (recall that they can be equivalently replaced by non-answer variables, Section 2). Also, assume that $q, q_1, q_2, q_{\texttt{lgg}}$ heads are $q(\bar{x}), q_1(\bar{x_1}), q_2(\bar{x_2}), q_{\texttt{lgg}}(\bar{x}_{\texttt{lgg}})$ respectively.

To prove our claim, we need to show that $q \models_{\mathcal{R},\mathcal{O}} q_{\texttt{lgg}}$ holds. Since $q_{\texttt{lgg}}$ is an `lgg` of q_1 and q_2 w.r.t. $\mathcal{O}$, there exist two homomorphisms ϕ_1 and ϕ_2 from the variables in $q_{\texttt{lgg}}$ to the values (variables and constants) in $q_1{}^{\infty}_{\mathcal{O}}$ and in $q_2{}^{\infty}_{\mathcal{O}}$ respectively, such that $[body(q_{\texttt{lgg}})]_{\phi_1} \subseteq body(q_1{}^{\infty}_{\mathcal{O}})$ and $\phi_1(\bar{x}_{\texttt{lgg}}) = \bar{x_1}$, and similarly, $[body(q_{\texttt{lgg}})]_{\phi_2} \subseteq body(q_2{}^{\infty}_{\mathcal{O}})$ and $\phi_2(\bar{x}_{\texttt{lgg}}) = \bar{x_2}$. Consider the BGPQ $q'_{\texttt{lgg}}$ obtained from $q_{\texttt{lgg}}$ by replacing every variable v by $v_{t_1t_2}$ where $t_1 = \phi_1(v)$ and $t_2 = \phi_2(v)$. Clearly, $q_{\texttt{lgg}} \equiv_{\mathcal{R},\mathcal{O}} q'_{\texttt{lgg}}$ holds. Assume that the head of $q'_{\texttt{lgg}}$ is $q'_{\texttt{lgg}}(\bar{x}'_{\texttt{lgg}})$; by construction is has the same head as q. Let us show that $q \models_{\mathcal{R},\mathcal{O}} q'_{\texttt{lgg}}$ holds, i.e., there exists a homomorphism ϕ from the variables in $q'_{\texttt{lgg}}$ to values in $q^{\infty}_{\mathcal{O}}$ such that $[body(q'_{\texttt{lgg}})]_{\phi} \subseteq body(q^{\infty}_{\mathcal{O}})$ and $\phi(\bar{x'}_{\texttt{lgg}}) = \bar{x}$. By Definition of q, it is easy to see that the above holds when ϕ maps $v_{t_1t_2}$ either to t_1 if $t_1 = t_2$ and $t_1 \in \mathcal{U} \cup \mathcal{L}$, or to itself otherwise. Importantly, this entails (i) the existence of the cover query q when an `lgg` $q_{\texttt{lgg}}$ exists and (ii) that $q \models_{\mathcal{R},\mathcal{O}} q'_{\texttt{lgg}}$ holds, hence $q \models_{\mathcal{R},\mathcal{O}} q_{\texttt{lgg}}$ holds, and therefore q is an `lgg` of q_1 and q_2 w.r.t. $\mathcal{O}$. □

As an immediate consequence of the above results, we get the following worst-case time to compute an `lgg` of two BGPQs q_1 and q_2, and its size. We assume given the saturation $q_1{}^{\infty}_{\mathcal{O}}$ and $q_2{}^{\infty}_{\mathcal{O}}$ w.r.t. the sets $\mathcal{O}$ of ontological constraints and $\mathcal{R}$ of RDF entailment rules under consideration, as the times to compute $q_1{}^{\infty}_{\mathcal{O}}$ and $q_2{}^{\infty}_{\mathcal{O}}$, and their sizes, depend on the particular $\mathcal{O}$ and $\mathcal{R}$ at hand.

COROLLARY 2. *An* `lgg` *of two BGPQs q_1 and q_2 can be computed in $O(|body(q_1{}^{\infty}_{\mathcal{O}})| \times |body(q_2{}^{\infty}_{\mathcal{O}})|)$ and its size is $|body(q_1{}^{\infty}_{\mathcal{O}})| \times |body(q_2{}^{\infty}_{\mathcal{O}})|$.*

Similarly to the preceding section, computing naively the cover query-based `lgg` of n BGPQs of size M based on Proposition 3 leads to an `lgg` of size M^n. However, remov-

ing the redundant triples from the intermediate and final cover query-based lggs limits their size to at most M.

Figure 8 (right) displays the cover query of the BGPQs $q_1^{\infty}_{\mathcal{O}}$ and $q_2^{\infty}_{\mathcal{O}}$ shown in Figure 7. It is therefore (Theorem 8) an lgg of the BGPQs q_1 and q_2 shown in Figure 6 w.r.t. the ontological constraints shown in Figure 7, using the RDF entailment rules shown in Table 2.

Figure 8 exemplifies the benefits of taking into account extra ontological constraints modeling background knowledge when identifying the commonalities between queries, thus of endowing the RDF relation of generalization/specialization between queries with such knowledge. When background knowledge is ignored (left), we only learn that both q_1 and q_2 ask for *the resources having some type*. In contrast, when we do consider background knowledge (right), we further learn that these resources, which both q_1 and q_2 ask for, are *publications, which have some researcher as author*.

5. ALGORITHMS

We provide algorithms to compute lggs of RDF graphs and BGPQs, based on the results obtained in the preceding Sections. In Section 5.1, we present an algorithm for computing the least general anti-unifiers of tuples of RDF values (URIs, literals, blank nodes and variables), i.e., of triples, triple patterns or query heads, on the definitions of cover graphs and queries rely. Then, in Section 5.2, we give three algorithms to compute a cover graph-based lgg of RDF graphs when the input and output RDF graphs fit in memory, in data management systems or in MapReduce clusters. Finally, in Section 5.3, we provide an algorithm to compute a cover query-based lgg of BGPQs.

5.1 Least general anti-unification

Algorithm 1, called lgau, computes a least general anti-unifier $(t_1^T, \ldots, t_n^T)$ of two n-ary tuples of RDF values $(t_1, \ldots, t_n)$ and $(t'_1, \ldots, t'_n)$, made of constants (URIs and literals), blank nodes and variables. This is achieved by setting the i^{th} value t_i^T of the output tuple to the least general generalization of the values found at the i^{th} positions of the two input tuples: t_i and t'_i. Recall that a pair of a same constant is generalized by that constant itself, while in all other cases the generalization is either a blank node in case of RDF graph triple values (Section 3.2) or a variable in case of query head answer variables or of query body triple pattern values (Section 4.2). Generalizing pairs of differents values with blank nodes or variables is controlled with the Boolean *bnodes* parameter. Crucially, these generated blank nodes or variables adopt the consistent naming scheme devised in Sections 3.2 and 4.2 that allows us preserving the common input RDF graphs (resp. BGPQs) structure across the anti-unifications of triples (resp. triple patterns).

5.2 lggs of RDF graphs

Following Definition 2, Algorithm 2, called lgg4g, computes the cover graph $\mathcal{G}$ of two input RDF graphs $\mathcal{G}_1$ and $\mathcal{G}_2$: $\mathcal{G}$ comprises the least general anti-unification of every pair of $\mathcal{G}_1$ and $\mathcal{G}_2$ triples with *same* property. Therefore, given two RDF graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, a call lgg4g$(\mathcal{G}_1, \mathcal{G}_2)$ produces the cover graph-based lgg of $\mathcal{G}_1$ and $\mathcal{G}_2$ ignoring RDF entailment (Theorem 2), while a call lgg4g$(\mathcal{G}_1^{\infty}, \mathcal{G}_2^{\infty})$ produces the cover graph-based lgg of $\mathcal{G}_1$ and $\mathcal{G}_2$ taking into account the set of RDF entailment rules at hand (Theorem 3). In the latter case, the input RDF graphs can be saturated using standard algorithms implemented in RDF data management systems, like Jena [3] and Virtuoso [8].

Algorithm 1 Least general anti-unification: lgau

In: $T_1 = (t_1, \ldots, t_n)$ and $T_2 = (t'_1, \ldots, t'_n)$, boolean *bnodes*
Out: least general anti-unification T of T_1 and T_2

```
for i = 1 to n do                          ▷ for each pair of ith attributes
  if t_i = t'_i and t_i ∈ U ∪ L then
    t_i^T ← t_i                            ▷ generalization of a constant by itself
  else if bnodes then                      ▷ cover graph case
    t_i^T ← b_{t_i t'_i}                   ▷ generalization by a blank node
  else                                     ▷ cover query case
    t_i^T ← v_{t_i t'_i}                   ▷ generalization by a variable
  end if
end for
return (t_1^T, ..., t_n^T)                 ▷ () when n = 0
```

Algorithm 2 Cover graph of two RDF graphs: lgg4g

In: RDF graphs $\mathcal{G}_1$ and $\mathcal{G}_2$
Out: $\mathcal{G}$ is the cover graph of $\mathcal{G}_1$ and $\mathcal{G}_2$

```
G ← ∅
for all T_1 = (s_1, p_1, o_1) ∈ G_1 do
  for all T_2 = (s_2, p_2, o_2) ∈ G_2 with p_1 = p_2 do
    G ← G ∪ {lgau(T_1, T_2, true)}
  end for
end for
return G
```

Importantly, lgg4g assumes that the input RDF graphs, as well as their output cover graph, fit in memory. Checking whether this is the case for the input RDF graphs under consideration can be done as follows.

The size of the input RDF graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ can be computed with the following SPARQL queries counting how many triples each of them holds: `SELECT count(*) as` ?*size* `FROM` $\mathcal{G}_i$ with $i \in [1, 2]$. Recall that the worst-case size of the output cover graph is $|\mathcal{G}| = |\mathcal{G}_1| \times |\mathcal{G}_2|$ in the unlikely case where *all* the $\mathcal{G}_1$ and $\mathcal{G}_2$ triples use the same property (Property 2 and Corollary 1).

The precise size of the output cover graph $\mathcal{G}$ can be computed, *without* computing $\mathcal{G}$, with SPARQL queries. First, we calculate for each input RDF graph $\mathcal{G}_i$, with $i \in [1, 2]$, how many triples it holds per distinct property p:

$$S_{\mathcal{G}_i} = \{(p, n_i) \mid |\{(\mathtt{s}, p, \mathtt{o}) \in \mathcal{G}_i\}| = n_i\}$$

This can be computed with the SPARQL query: `SELECT` ?p `count(*) as` ?n_i `FROM` $\mathcal{G}_i$ `WHERE` $\{(?s, ?p, ?o)\}$ `GROUP BY` ?p. Then, since every $\mathcal{G}_1$ triple with property p anti-unifies with every $\mathcal{G}_2$ triple with same property p, in order to beget $\mathcal{G}$, the size of $\mathcal{G}$ is:

$$|\mathcal{G}| = \sum_{(p,n_1) \in S_{\mathcal{G}_1}, (p,n_2) \in S_{\mathcal{G}_2}} n_1 \times n_2$$

Overall, $|\mathcal{G}|$ can be computed with the SPARQL query: `SELECT SUM(`?n_1`*`?n_2`) as` ?*size* `WHERE` $\{\{S_{\mathcal{G}_1}\}\{S_{\mathcal{G}_2}\}\}$ with $S_{\mathcal{G}_1}$ and $S_{\mathcal{G}_2}$ denoting the above-mentioned SPARQL queries computing these two sets, which join on their commun answer variable ?p.

When the input RDF graphs or their output cover graph cannot fit in memory, we propose variants of lgg4g that either assume that RDF graphs are stored in data manage-

Algorithm 3 Cover graph of two RDF graphs: `lgg4g-dms`

In: cursor c_1 on RDF graph $\mathcal{G}_1$, cursor c_2 on RDF graph $\mathcal{G}_2$, write access to an empty RDF graph $\mathcal{G}$, integer n
Out: $\mathcal{G}$ is the cover graph of $\mathcal{G}_1$ and $\mathcal{G}_2$

1: c_1.`init`() ▷ *c_1 at beginning of $\mathcal{G}_1$*
2: **while** $B_1 = c_1$.`next`(n) **do** ▷ *fetch next n $\mathcal{G}_1$ triples*
3: c_2.`init`() ▷ *c_2 at beginning of $\mathcal{G}_2$*
4: **while** $B_2 = c_2$.`next`(n) **do** ▷ *fetch next n $\mathcal{G}_2$ triples*
5: **for all** $T_1 = (\mathtt{s}_1, \mathtt{p}_1, \mathtt{o}_1) \in B_1$ **do**
6: **for all** $T_2 = (\mathtt{s}_2, \mathtt{p}_2, \mathtt{o}_2) \in B_2$ with $\mathtt{p}_1 = \mathtt{p}_2$ **do**
7: insert `lgau`(T_1, T_2, true) into $\mathcal{G}$
8: **end for**
9: **end for**
10: **end while**
11: **end while**

Algorithm 4 Cover graph of two RDF graphs: `lgg4g-mr`

In: file G_1 for RDF graph $\mathcal{G}_1$, file G_2 for RDF graph $\mathcal{G}_2$
Out: $\mathcal{G}$ is the cover graph of $\mathcal{G}_1$ and $\mathcal{G}_2$, stored in G-$*$ files

`Map`(key: file G_i, value: triple $T_i = (\mathtt{s}_i, \mathtt{p}_i, \mathtt{o}_i)$)
1: `emit`$(\langle \mathtt{p}_i, (G_i, T_i) \rangle)$

`Reduce`(key: p, values: set $\mathcal{V}$ of values emitted for key p)
1: $f \leftarrow$ `open`$(G$-$p)$
2: **for all** $(G_1, T_1 = (\mathtt{s}_1, p, \mathtt{o}_1)) \in \mathcal{V}$ **do**
3: **for all** $(G_2, T_2 = (\mathtt{s}_2, p, \mathtt{o}_2)) \in \mathcal{V}$ **do**
4: $f.write($`lgau`$(T_1, T_2, \text{true}))$
5: **end for**
6: **end for**
7: `close`(f)

ment systems (DMSs, in short) or in a MapReduce cluster.

5.2.1 Handling large RDF graphs using DMSs

Algorithm 3, called `lgg4g-dms`, is an adaptation of `lgg4g`, which assumes that the input RDF graphs (already saturated if needed) and their cover graph are all stored in one or several DMSs. It further assumes that the system(s) storing the input RDF graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ feature(s) the well-known database mechanism of *cursor* [22, 41]. This is for instance the case for RDF graphs stored in relational servers like DB2 [1], MySQL [4], Oracle [5] and PostgreSQL [6], or in RDF servers like Jena-TDB [3] and Virtuoso [8]. Roughly speaking, a cursor is a pointer or iterator on tuples hold in a DMS, e.g., stored a relation or computed as the results to a query, that can be used to access these tuples. In particular, a cursor can be used by an application to iteratively traverse all the tuples by fetching n of them at a time.

`lgg4g-dms` uses cursors to proceed similarly to `lgg4g` (remark that lines 5-9 in Algorithm 3 are almost the same as lines 2-6 in Algorithm 2) on pairs of n-triples subsets of $\mathcal{G}_1$ and of $\mathcal{G}_2$, instead of on the whole RDF graphs themselves. It follows that the worst-case number of triples kept in memory by `lgg4g-dms` is $M = (2 \times n) + 1$ at line 7 (i.e., B_1, B_2 and the anti-unification triple output by `lgau`), with:

$$3 \leq M \leq |\mathcal{G}_1| + |\mathcal{G}_2| + 1.$$

The above lower bound is met for n set to 1, while the upper one is met for n set to $\max(|\mathcal{G}_1|, |\mathcal{G}_2|)$. Importantly, `lgg4g-dms` allows *choosing* the value of n in order to reflect the memory devoted to handling triples. For instance, if one wants to use 4GB of RAM for triples, assuming that any triple fits in less one 1KB (this value is much less when using dictionary encoding), the value of n can be set to 2M.

This clearly contrasts with the worst-case number of triples kept in memory by `lgg4g`: $M = |\mathcal{G}_1| + |\mathcal{G}_2| + |\mathcal{G}|$ at line 7, with:

$$|\mathcal{G}_1| + |\mathcal{G}_2| \leq M \leq |\mathcal{G}_1| + |\mathcal{G}_2| + (|\mathcal{G}_1| \times |\mathcal{G}_2|).$$

The above lower bound is met when $\mathcal{G}_1$ and $\mathcal{G}_2$ have no property in common in their triples (i.e., $|\mathcal{G}| = 0$), while the upper one is met in the unlikely case where $\mathcal{G}_1$ and $\mathcal{G}_2$ use a same property in all their triples (i.e., $|\mathcal{G}| = |\mathcal{G}_1| \times |\mathcal{G}_2|$).

5.2.2 Handling huge RDF graphs using MapReduce

Algorithm 4, called `lgg4g-mr`, is a MapReduce (MR) variant of `lgg4g`. MR is a popular massively parallel programming framework [21], implemented by many large-scale data processing systems, like Hadoop [2] and Spark [7], which orchestrate clusters of compute nodes.

A MR program is organized in successive *jobs*, each of which comprises a *Map task* followed by a *Reduce task*. The Map task consists in reading some input data from the distributed file system[2] of the cluster, so as to partition the data into $\langle \mathtt{k}, \mathtt{v} \rangle$ key-value pairs. Importantly, an MR engine transparently processes the Map task by running *Mapper processes* in parallel on cluster nodes, each process taking care of partitioning a portion of the input data by applying a `Map`(key: file, value: data unit) function on every data unit of a given input file. Key-value pairs thus produced are shuffled across the network, so that *all* pairs with *same* key $\langle \mathtt{k}, \mathtt{v}_1 \rangle \cdots \langle \mathtt{k}, \mathtt{v}_n \rangle$ are shipped to a same compute node. The Reduce task then consists in running *Reducer processes* in parallel, for every distinct key $\mathtt{k}$ received by every compute node. Each process takes care of the set $\mathcal{V}$ of values $\{\mathtt{v}_1, \ldots, \mathtt{v}_n\}$ emitted with key $\mathtt{k}$, by applying a `Reduce`(key: k, values: $\mathcal{V}$) function, and writing its results in a file. The result of an MR job, comprises the data, stored in a distributed fashion, in all the files output by Reducers.

In `lgg4g-mr`, the `Map` function applies to every $(\mathtt{s}_i, \mathtt{p}_i, \mathtt{o}_i)$ triple of the input RDF graph $\mathcal{G}_i$ stored in file G_i, and produces the corresponding key-value pair $\langle \mathtt{p}_i, (G_i, (\mathtt{s}_i, \mathtt{p}_i, \mathtt{o}_i)) \rangle$, for $i \in [1, 2]$. Hence, all the $\mathcal{G}_1$ and $\mathcal{G}_2$ triples with a same key/property p are shipped to the same cluster node. Then, similarly to `lgg4g` at lines 2-6, the `Reduce` functions process, on each node, the set $\mathcal{V}$ of values emitted for every received key p. The least general anti-unification triples obtained at line 4 are stored in the output file G-p. At the end of the MR job, the `lgg` $\mathcal{G}$ of $\mathcal{G}_1$ and $\mathcal{G}_2$ is stored in the G-$*$ files of the distributed file system, where $*$ denotes any key/property p.

In `lgg4g-mr`, the `Map` function holds at most a single $\mathcal{G}_1$ or $\mathcal{G}_2$ triple in memory. In constrast, the worst-case number of triples handled by the `Reduce` function for a given key p is: $M = |\mathcal{G}_1| + |\mathcal{G}_2| + 1$ at line 4. This upper bound is met in the unlikely case where $\mathcal{G}_1$ and $\mathcal{G}_2$ use the same property p in all their triples. Similarly to `lgg4g-dms`, this upper bound can set to $M = (2 \times n) + 1$, with $3 \leq M \leq |\mathcal{G}_1| + |\mathcal{G}_2| + 1$, by first splitting the input RDF graphs in k_i files of n $\mathcal{G}_i$ triples (files $G_i^1, \ldots, G_i^{k_i}$), and then by processing every pair of n-triples of $\mathcal{G}_1$ and $\mathcal{G}_2$ files with an MR job (i.e., with $k_1 \times k_2$ jobs), instead of a single MR job for the entire two

[2]For simplicity, we assume that input and output data of an MR job is stored on disk, like in Hadoop, while it may also reside in in-memory shared data structures, like in Spark.

Algorithm 5 Cover query of two BGPQs: `lgg4q`

In: BGPQs q_1 and q_2
Out: Cover query q of q_1 and q_2 if it exists, else $\perp$
1: $body(q) \leftarrow \emptyset$
2: **for all** $T_1 = (\mathtt{s}_1, \mathtt{p}_1, \mathtt{o}_1) \in \mathcal{G}_1$ **do**
3: **for all** $T_2 = (\mathtt{s}_2, \mathtt{p}_2, \mathtt{o}_2) \in \mathcal{G}_2$ **do**
4: $body(q) \leftarrow body(q) \cup \{\mathtt{lgau}(T_1, T_2, \text{false})\}$
5: **end for**
6: **end for**
7: $head(q) \leftarrow \mathtt{lgau}(head(q_1), head(q_2), \text{false})$
8: **if** q is a well-formed BGPQ **then**
9: **return** q
10: **else**
11: **return** $\perp$ ▷ error: a cover query does not exist
12: **end if**

input RDF graphs.

Finally, to take into account RDF entailment, the input RDF graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ can be saturated before being stored in the MR cluster, by using standard (centralized) techniques, or within the MR cluster, by using MR-based saturation techniques [45, 44]. Also, it is worth noting that RDF graphs, thus `lgg`s of RDF graphs, stored in an MR cluster can be queried with MR-based SPARQL engines [27, 30, 35, 23].

5.3 `lgg`s of BGPQs

Following Definition 6, Algorithm 5, called `lgg4q`, builds and returns either the cover query q of two input BGPQs q_1 and q_2 if it exists, or the $\perp$ symbol otherwise.

At lines 2-6, `lgg4q` computes the body of q, which comprises a least general anti-unifier of every pair of q_1 body and q_2 body triples (first item in Definition 6). Then, at line 7, it computes the head of q as the least general anti-unifier of the heads of q_1 and q_2 (second item in Definition 6). Finally, `lgg4q` checks whether q is a well-formed BGPQ, i.e., if its answer variables occur in its body triples, otherwise there is no cover query of q_1 and q_2, and also no `lgg` of q_1 and q_2 (recall first item of Theorems 7 and 8).

Therefore, given two BGPQs q_1 and q_2, a call $\mathtt{lgg4q}(q_1, q_2)$ produces the cover query-based `lgg` of q_1 and q_2 ignoring RDF entailment and extra ontological constraints, whenever it exists (Theorem 7), while a call $\mathtt{lgg4q}(q_1^{\infty_\mathcal{O}}, q_2^{\infty_\mathcal{O}})$ produces the cover query-based `lgg` of q_1 and q_2 taking into account the set of RDF entailment rules at hand, as well as a set of extra ontological constraints, whenever it exists (Theorem 8). In the latter case, the saturated input BGPQs can be computed using Algorithm 6, called `sat4q`.

Following Definition 3, `sat4q` computes the saturation of a query q w.r.t. a set $\mathcal{R}$ of RDF entailment rules and a set $\mathcal{O}$ of RDFS constraints as a BGPQ $q_\mathcal{O}^\infty$ with same head as q (line 1), and whose body is computed based on its characterization displayed in Figure 5: $body(q_\mathcal{O}^\infty) = body(q)^\infty \cup ((body(q) \cup \mathcal{O})^\infty \setminus \mathcal{O}^\infty)$ (*). Further, since $(body(q) \cup \mathcal{O})^\infty$ is equivalent to $(body(q)^\infty \cup \mathcal{O}^\infty)^\infty$ (because $body(q)$ and $\mathcal{O}$ are equivalent w.r.t. $\mathcal{R}$ to $body(q)^\infty$ and $\mathcal{O}^\infty$ respectively), $body(q_\mathcal{O}^\infty) = body(q)^\infty \cup ((body(q)^\infty \cup \mathcal{O}^\infty)^\infty \setminus \mathcal{O}^\infty)$ (**) holds. `sat4q` uses (**) instead of (*) since, *once* $body(q)^\infty$ and $\mathcal{O}^\infty$ are computed (lines 2 and 3), clearly $(body(q)^\infty \cup \mathcal{O}^\infty)^\infty$ is faster to compute than $(body(q) \cup \mathcal{O})^\infty$, thus (**) is faster to compute (line 5) than (*).

Algorithm 6 BGPQ saturation w.r.t. constraints: `sat4q`

In: BGPQ q, set $\mathcal{R}$ of RDF entailment rules and set $\mathcal{O}$ of RDFS statements
Out: $q_\mathcal{O}^\infty$ is the saturation of q w.r.t. $\mathcal{R}$ and $\mathcal{O}$
1: $head(q_\mathcal{O}^\infty) \leftarrow head(q)$
2: $\mathcal{O}_s \leftarrow \mathcal{O}^\infty$
3: $bq_s \leftarrow body(q)^\infty$
4: rename bq_s blank nodes shared with $\mathcal{O}_s$ ▷ *they are local*
5: $body(q_\mathcal{O}^\infty) \leftarrow bq_s \cup ((bq_s \cup \mathcal{O}_s)^\infty \setminus \mathcal{O}_s)$
6: **return** $q_\mathcal{O}^\infty$

6. EXPERIMENTS

We provide an experimental assessment of our technical contribution for computing `lgg`s of BGPQs, which goes beyond that for RDF graphs (recall that BGPQ bodies generalize RDF graphs). We experimentally study the added-value of considering background knowledge when learning `lgg`s of queries. As our next result shows (Proposition 5), this amounts to measuring *how more specific (pregnant)* is an `lgg` of queries that considers background knowledge *than* an `lgg` of the same queries that ignores background knowledge:

PROPOSITION 5. *Let $\mathcal{R}$ be a set of RDF entailment rules, $\mathcal{O}$ a set of RDFS statements, and q_1, q_2 two BGPQs with same arity. An* `lgg` $q_{1\mathtt{gg}}$ *of q_1, q_2 and an* `lgg` $q_{1\mathtt{gg}}^\mathcal{O}$ *of q_1, q_2 w.r.t. $\mathcal{O}$ (Definition 5) are such that: $q_{1\mathtt{gg}}^\mathcal{O} \models_\mathcal{R} q_{1\mathtt{gg}}$ holds.*

PROOF. Since the `lgg`s of some queries are equivalent (recall that the `lgg` is semantically unique): (i) $q_{1\mathtt{gg}}$ is *equivalent* to the cover query-based `lgg` q of the saturations with $\mathcal{R}$ of q_1 and of q_2 w.r.t. the *empty set* of RDFS constraints, and (ii) $q_{1\mathtt{gg}}^\mathcal{O}$ is *equivalent* to the cover query-based `lgg` q' of the saturations with $\mathcal{R}$ of q_1 and of q_2 w.r.t. $\mathcal{O}$. Further, *by definition* of a cover query (Definition 6), q and q' have the same heads and the body of q is a subset of that q', thus (iii) $q' \models_\mathcal{R} q$ holds. Therefore, from (i), (ii) and (iii), $q_{1\mathtt{gg}}^\mathcal{O} \models_\mathcal{R} q_{1\mathtt{gg}}$ holds. □

Based on the above result, as a practical metric for measuring the semantic distance between $q_{1\mathtt{gg}}^\mathcal{O}$ and $q_{1\mathtt{gg}}$ through $\models_\mathcal{R}$, and because `lgg`s are BGPQs, we compute the *gain in precision* background knowledge yields *w.r.t. query answering*. For an RDF graph $\mathcal{G}$ with set $\mathcal{O}$ of RDFS statements, we have:

$$\text{gain in precision (in \%)} = 1 - \frac{|q_{1\mathtt{gg}}^\mathcal{O}(\mathcal{G}) \cap q_{1\mathtt{gg}}(\mathcal{G})|}{|q_{1\mathtt{gg}}(\mathcal{G})|} = 1 - \frac{|q_{1\mathtt{gg}}^\mathcal{O}(\mathcal{G})|}{|q_{1\mathtt{gg}}(\mathcal{G})|}$$

since $q_{1\mathtt{gg}}^\mathcal{O} \models_\mathcal{R} q_{1\mathtt{gg}}$ holds (Proposition 5), hence $q_{1\mathtt{gg}}^\mathcal{O}(\mathcal{G}) \subseteq q_{1\mathtt{gg}}(\mathcal{G})$ holds.

In Section 6.1, we describe our experimental settings. Then, in Section 6.2, we study the saturation of a BGPQ w.r.t. ontological constraints. Finally, in Section 6.3, we study the computation of `lgg`s with or without considering constraints, notably the gain in precision when constraints are considered.

6.1 Settings

Software. We implemented our technical contributions in Java 1.8, on top of the Jena 3.0.1 RDF reasoner and of a PostgreSQL 9.3.11 server, all used with default settings.

• We used Jena to compute the *saturation of an RDF graph*, against which queries must be evaluated to obtained their *complete* answer sets (Section 2); we also used Jena

LUBM query $Q_{1\leq i\leq 9}$:	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9
Q_i's shape	star	star	graph	graph	star	tree	graph	star	star
$\|body(Q_i)\|$	3	3	4	3	2	4	6	2	3
number of URI/variable occurrence in Q	5/4	5/4	5/7	4/5	3/3	6/6	7/11	2/4	5/4
$\|Q_i(\mathcal{G}_{\texttt{LUBM}})\|$	123	41	869	0	269	41 751	79	14 252	16
$\|body(Q_i^{\infty}{}_{\mathcal{O}_{\texttt{LUBM}}})\|$	8	9	11	7	6	8	14	10	8
Time to compute $Q_i^{\infty}{}_{\mathcal{O}_{\texttt{LUBM}}}$	24	23	23	21	22	22	21	27	22

Table 3: Characteristics of our test queries (top) and of their saturations w.r.t. LUBM constraints (bottom); times are in ms.

`lgg` of the LUBM queries:	Q_1Q_4	Q_2Q_4	Q_1Q_8	Q_2Q_8	Q_5Q_8	Q_1Q_9	Q_8Q_9	Q_3Q_6	Q_3Q_7
Time to compute $q_{\texttt{lgg}}$	1	2	1	1	1	3	2	2	4
$\|q_{\texttt{lgg}}(\mathcal{G}_{\texttt{LUBM}})\|$	30 963	1 048 360	1 048 360	74 643	1 048 360	253 443	1 048 360	6 937 472	57 774
Time to compute $q_{\texttt{lgg}}^{\mathcal{O}_{\texttt{LUBM}}}$	5	6	6	7	10	6	6	6	8
$\|q_{\texttt{lgg}}^{\mathcal{O}_{\texttt{LUBM}}}(\mathcal{G}_{\texttt{LUBM}})\|$	14 285	14 285	56 358	56 358	185 124	19 053	56 358	520 365	34 852
Gain in precision	53.86	98.63	94.62	24.49	82.34	92.48	94.62	92.49	39.67

Table 4: Characteristics of cover query-based `lgg`s of test queries, w or w/o using the LUBM constraints; times are in ms.

to compute the *saturation $q_{\mathcal{O}}^{\infty}$ of a BGPQ q w.r.t. a set $\mathcal{O}$ of RDFS constraints* (Definition 3, Algorithm 6): we rely on Jena's saturation, union and difference operators to compute $q_{\mathcal{O}}^{\infty}$'s body.

• We used PostgreSQL to evaluate *SQLized* BGPQs against a *saturated* RDF graph stored in a `Triple(s,p,o)` table. The table is indexed by all permutations of the `s`, `p`, `o` columns, leading to a total of 6 indexes. This indexing choice is inspired by [34, 49], to give PostgreSQL efficient query evaluation opportunities. There exists alternative data layouts like having one two-columns table `p(s,o)` per distinct property `p` in the RDF graph, which stores all the subject/object tuples of triples with property `p` [14], or the elaborate layout of [12] used in DB2 RDF. However, adopting another data layout than the `Triple` table would have no impact on our experiments, as the reported times are not related to query evaluation.

RDF entailment rules. We used the subset of standard RDF entailment rules in Table 2, which fully allows exploiting RDFS ontological constraints, i.e., background knowledge.

Datasets. We conducted experiments using *real DBpedia data* [32] and *synthetic LUBM data* [26].

• **LUBM dataset**: we generated an RDF graph $\mathcal{G}_{\texttt{LUBM}}$ comprising 884k triples before saturation, including a subset $\mathcal{O}_{\texttt{LUBM}}$ of 242 RDFS constraints. The saturation of $\mathcal{G}_{\texttt{LUBM}}$ comprises 1.08M triples after saturation.

• **DBpedia dataset**: we picked four complementary files[3] to build the RDF graph $\mathcal{G}_{\texttt{DBpedia}}$ comprising 41.18M triples, whose subset $\mathcal{O}_{\texttt{DBpedia}}$ of 30.31k RDFS constraints represents DBpedia's background knowledge. The saturation of $\mathcal{G}_{\texttt{DBpedia}}$ comprises 78.14M triples.

Queries. We used the following queries:

• **LUBM queries**: we borrowed from [13] a set of 9 BGPQs. They can be found in Appendix A. Their characteristics are displayed in Table 3 (top) : LUBM queries have a variety of structural aspects and numbers of answers.

[3]We use the `dbpedia_2015-10.nt` RDF Schema file and the `instance_types_en.ttl`, `mappingbased_literals_en.ttl` and `mappingbased_objects_en.ttl` RDF data files.

• **DBpedia queries**: we defined 42 test BGPQs, among we picked 8 representative ones with 2 variables. These 8 queries can be found in Appendix B. Their characteristics are displayed in Table 5 (top). Also, importantly, queries Q_1-Q_4, in Table 5 (left), are heterogeneous in the sense that they differ significantly both on their structure and the kind of information they ask for, hence use many distinct classes, properties and URI values, while Q_4-Q_8, in Table 5 (right), are homogeneous and only differ in some classes, properties and URI values.

Hardware. We used an Intel Xeon (X5550) 2.67GHz machine with 32GB RAM, using Ubuntu 14.04.3 LTS (64bits).

In the sequel, all measured times are averaged over 5 warm runs and are in milliseconds.

6.2 BGPQ saturation w.r.t. constraints

Tables 3 (bottom) and 5 (bottom) show the size of our saturated test queries and the time to compute them. Enriching our test queries using the LUBM and DBpedia constraints augments their size: from ×2 for Q_{10} up to ×5 for Q_{22} in the LUBM case; from ×3.16 for Q_2 up to ×4.75 for Q_3 in the DBpedia case. The query saturation time is rather fast with LUBM, 23ms on average, while it is 692ms on average with DBpedia. Saturation times have been multiplied by 30 from LUBM to DBPedia, since the latter has ×721 more constraints than the former.

6.3 `lgg`s of BGPQs w.r.t. constraints

LUBM. Table 4 (lines 1 and 3) shows that cover query-based `lgg`s of test queries are always computed fast whether or not we consider the LUBM constraints: 1 to 4ms when they are ignored, and 5 to 10ms when they are considered. In the latter case, overall, it takes between 50 and 59ms to compute an `lgg` in the worst case (i.e., when the two saturated test queries are computed *in sequence* before computing their cover query). Table 4 (lines 2 and 4) shows that ignoring extra constraints significantly increases the number of answers for some `lgg`s, from a small ×1.32 for Q_2Q_8 up to a striking ×73.39 for Q_2Q_4, with a significant average

of $\times$16.45. This translates into the precision gains shown at line 5 (Table 4): 74.80% overall, 98.63% for queries Q_2 and Q_4, and 24.49% for Q_2 and Q_8. lggs with same number of answers in Table 4 were found either equivalent or almost equivalent so that their semantic difference is not visible on the LUBM dataset, e.g., the lggs of Q_8Q_9, Q_5Q_8 and Q_1Q_8 with 1 048 360 answers (i.e., ignoring ontological constraints) are asking for all the (distinct) subject/object pairs in triples in the LUBM RDF graph; the lggs of Q_8Q_9, Q_2Q_8 with 56 358 answers (i.e., considering ontological constraints) ask respectively for faculties with some degree from some university and for employees with some doctoral degree from some university: they have the same number of answers in our LUBM RDF graph because all faculties have some doctoral degree and all employees with some doctoral degree are faculties.

DBpedia. First, as Table 6 (lines 1 and 3) shows, the cover query-based lggs of test queries are always computed fast whether or not the DBpedia constraints are considered: from 3 to 6ms when ignored, to 13 to 18ms when considered. In the latter case, overall, it takes between 1.322 and 1.457s to compute an lgg in the worst case, i.e., when the two saturated test queries are computed *in sequence* before computing their cover query.

Table 6 (lines 2 and 4) also shows that the answer set of an lgg is significantly larger when DBpedia constraints are not taken into account: the size difference goes from a small $\times$1.02 for the homogeneous queries Q_5, Q_7 up to a striking $\times$76.42 for the heterogeneous queries Q_1, Q_4, with a significant average of $\times$17.38 ($\times$33.34 for the heterogeneous queries and $\times$1.42 for the homogeneous ones). This translates into the precision gains shown at line 5: 57.78% overall, 90.18% for the heterogeneous queries, and 25.37% for the homogeneous ones.

Conclusions: Our results confirm our claim that *taking into account background knowledge yields more pregnant* lggs. Indeed, ontological constraints help finding *common super-* classes and properties to be used in lggs in place of the different ones used in input queries; when constraints are ignored, these can just be generalized using *variables*. Therefore, the more heterogeneous input queries are, the more such common super-classes and properties are used in their lgg instead of variables, and the more the gain in precision of their lgg is high. For homogeneous input queries, while less striking, the gain in precision is significant in general, as our DBpedia experiments show.

7. RELATED WORK

The problem of computing an lgg was introduced in the early 70's by G. Plotkin [39, 40] to generalize First Order Logic clauses w.r.t. θ-subsumption, a non-standard logical implication typical of Machine Learning. This problem has also been investigated in Knowledge Representation, for formalisms whose expressivity overlaps with our RDF setting, notably Description Logics (DLs) [29, 11, 50] and Conceptual Graphs (CGs) [17]. Finally, recently, this problem has started receiving attention in the Semantic Web field [20, 19, 31].

In DLs, computing an lgg of concepts (formulae) has been studied for $\mathcal{EL}$ and extensions thereof [29, 11, 50]. The $\mathcal{EL}$ setting translates into *particular tree-shaped* RDF graphs, which may feature RDFS subclass and domain constraints, and for which RDF entailment is limited to the use of these two constraints only[4]. In these equivalent RDF and $\mathcal{EL}$ fragments, the $\mathcal{EL}$ technique that computes an lgg of $\mathcal{EL}$ concepts, which is an $\mathcal{EL}$ concept, provides *only a (non least general) generalization* of their corresponding tree-shaped RDF graphs w.r.t. the problem we study: the (minimal) cover graph-based lgg of tree-shaped RDF graphs is clearly a forest-shaped RDF graph in general. However, it can be shown that our technique for general RDF graph can be used to compute the $\mathcal{EL}$ lgg of $\mathcal{EL}$ concepts as shown in Example 1 below. Roughly speaking, given the $\mathcal{EL}$ concepts $C_1, \ldots, C_n$ and their corresponding tree-shaped RDF graphs $\mathcal{G}(C_1, b_1), \ldots, \mathcal{G}(C_n, b_n)$, the $\mathcal{EL}$ lgg is that corresponding to the tree-shaped RDF graph rooted in $b_{b_1,\ldots,b_n}$ within the forest-shaped cover graph of $\mathcal{G}(C_1, b_1), \ldots, \mathcal{G}(C_n, b_n)$.

In CGs, the so-called *simple CGs with unary and binary relations* correspond to *particular* RDF graphs (e.g., a property URI in a triple cannot be the subject or object of another triple, a class - URI or blank node - in a τ triple cannot be the subject of another τ triple nor the subject or object of another non-τ triple, etc), which may feature the four RDFS constraints, and for which RDF entailment is limited to the use of these RDFS contraints only [**?**]. In these equivalent RDF and CG fragments, we may interchangeably compute lggs with the CG technique in [17] or ours.

In RDF, computing an lgg has been studied for *particular* RDF graphs, called *r-graphs*, *ignoring RDF entailment* [20, 19]. An r-graph is an *extracted subgraph* of an RDF graph $\mathcal{G}$, *rooted* in the $\mathcal{G}$ value r and comprising the $\mathcal{G}$ triples *reachable from r through directed* paths of length at most n. Such a rooted and directed r-graph can be defined recursively as $\mathcal{S}(\mathcal{G}, r, n)$, with:

$$\mathcal{S}(\mathcal{G}, v, 0) = \emptyset$$

$$\mathcal{S}(\mathcal{G}, v, n) = \bigcup_{(v,p,v') \in \mathcal{G}} \{(v, p, v')\} \cup \mathcal{S}(\mathcal{G}, v', n-1) \cup \mathcal{S}(\mathcal{G}, p, n-1)$$

Intuitively, this purely structural definition of r-graph attempts carrying $\mathcal{G}$'s knowledge about r. lggs of r-graphs allow finding the commonalities between *single root entities*, while with general RDF graphs we further allow finding the commonalities between *sets of multiple interrelated entities*. The technique for computing an lgg of two r-graphs exploits their rooted and directed structure: it starts from their respective root and traverses them simultaneously considering triples reachable through directed paths of increasing size, while incrementally constructing an r-graph lgg. In contrast, the general RDF graphs we consider are unstructured; our technique blindly traverses the input RDF graphs to anti-unify their triples with same property, and captures their common structure across these anti-unifications thanks to the consistent naming scheme we devised for the blank nodes they generate. Further, when we ignore RDF entailment, computing lggs of r-graphs or of RDF graphs have the same worst-case time complexity ($O(|\mathcal{G}_1| \times |\mathcal{G}_2|)$, with $\mathcal{G}_1, \mathcal{G}_2$ the input RDF graphs). This is for instance the case

[4]An $\mathcal{EL}$ concept C recursively translates into the RDF graph rooted in the blank node b_r returned by the call $\mathcal{G}(C, b_r)$, with: $\mathcal{G}(\top, b) = \emptyset$ for the universal $\mathcal{EL}$ concept $\top$, $\mathcal{G}(A, b) = \{(b, \tau, A)\}$ for an atomic $\mathcal{EL}$ concept A, $\mathcal{G}(\exists r.C, b) = \{(b, r, b')\} \cup \mathcal{G}(C, b')$, with b' a fresh blank node, for an $\mathcal{EL}$ existential restriction $\exists r.C$, and $\mathcal{G}(C_1 \sqcap C_2, b) = \mathcal{G}(C_1, b) \cup \mathcal{G}(C_2, b)$ for an $\mathcal{EL}$ conjunction $C_1 \sqcap C_2$; the $\mathcal{EL}$ constraints $A_1 \sqsubseteq A_2$ and $\exists r.\top \sqsubseteq A$ correspond to $(A_1, \preceq_{\text{sc}}, A_2)$ and $(r, \hookleftarrow_d, A)$ resp.

DBpedia query $Q_{1\leq i\leq 8}$:	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8
Q_i's shape	tree	tree	tree	graph	graph	graph	graph	graph
$\|body(Q_i)\|$	4	6	4	6	4	6	6	6
Number of URI/variable occurrence in Q_i	7/5	9/9	5/7	7/11	5/7	9/9	9/9	9/9
$\|Q_i(\mathcal{G}_{\texttt{DBpedia}})\|$	77	0	41 695	13	6	0	1	0
$\|body(Q_i{}^{\infty}_{\mathcal{O}_{\texttt{DBpedia}}})\|$	16	19	19	23	16	23	23	23
Time to compute $Q_i{}^{\infty}_{\mathcal{O}_{\texttt{DBpedia}}}$	666	643	677	734	681	706	697	736

Table 5: Characteristics of our test BGPQs (top) and of their saturations w.r.t. DBpedia constraints (bottom); times are in ms.

`lgg` of the DBpedia queries:	Q_1Q_2	Q_1Q_3	Q_1Q_4	Q_2Q_3	Q_4Q_5	Q_5Q_6	Q_5Q_7	Q_7Q_8
Time to compute $q_{\texttt{lgg}}$	3	3	5	4	4	5	6	5
$\|q_{\texttt{lgg}}(\mathcal{G}_{\texttt{DBpedia}})\|$	477 455	34 747 102	34 901 117	60 356 807	1 977	1221	35	70
Time to compute $q_{\texttt{lgg}}^{\mathcal{O}_{\texttt{DBpedia}}}$	13	14	14	15	15	14	17	18
$\|q_{\texttt{lgg}}^{\mathcal{O}_{\texttt{DBpedia}}}(\mathcal{G}_{\texttt{DBpedia}})\|$	10 637	7 874 768	456 690	7 874 768	1 701	780	34	36
Gain in precision	97.77	77.33	98.69	86.95	13.96	36.11	2.85	48.57

Table 6: Characteristics of cover query-based `lgg`s of test queries, w/ or w/o using the DBpedia RDFS constraints; times are in ms.

for the following star-shaped RDF graphs, which are trees, r-graphs and RDF graphs, $\mathcal{G}_1 = \{(r_1, p, s_1^1), \ldots, (r_1, p, s_1^m)\}$ and $\mathcal{G}_2 = \{(r_2, p, s_2^1), \ldots, (r_2, p, s_2^n)\}$, the r-graph and cover graph-based `lgg` of which is built at some point by both techniques is: $\mathcal{G}_{\texttt{lgg}} = \{(b_{r_1r_2}, p, b_{s_1^1 s_2^1}), \ldots, (b_{r_1r_2}, p, b_{s_1^m s_2^n})\}$. Also, as noted in [19], the computed r-graphs `lgg`s are *only (non least general) generalizations* of r-graphs w.r.t. the standard semantics of RDF graphs defined upon RDF entailment. Finally, the r-graph technique that computes an `lgg` of r-graphs, which is an r-graph, gives *only a (non least general) generalization* of them w.r.t. the problem we study: the (minimal) cover graph-based `lgg` of r-graphs is clearly a general RDF graph as shown in Example 1 below.

In SPARQL, computing an `lgg` has been considered for *unary tree-shaped conjunctive queries (UTCQ)* [31]; a UTCQ `lgg` is computed by a simultaneous root-to-leaves traversal of the input queries. UTCQs are tree-shaped RDF graphs, when variables are viewed as blank nodes, for which RDF entailment is ignored [19]. The UTCQ technique that computes an `lgg` of UTCQs, which is a UTCQ, yields *only a (non least general) generalization* of their corresponding tree-shaped RDF graphs w.r.t. the problem we study: the (minimal) cover graph-based `lgg` of tree-shaped RDF graphs is clearly a forest-shaped RDF graph. However, similarly as for the $\mathcal{EL}$ description logic above, it can be shown that our technique for general RDF graph can be used to compute the UTCQ `lgg` of UTCQs as shown in Example 1 below.

Example 1. *Let us consider the two tree-shaped RDF graphs below that may correspond to $\mathcal{EL}$ concepts, r-graphs or UTCQs:*
$\mathcal{G}_1 = \{(b_1, p_e, b_{11}), (b_1, p_s, b_{12})\}$ *and*
$\mathcal{G}_2 = \{(b_2, p_e, b_{21}), (b_{21}, p_s, b_{211})\}$.

Their RDF graph `lgg` *is the general forest-shaped RDF graph* $\mathcal{G} = \{(b_{b_1b_2}, p_e, b_{b_{11}b_{21}}), (b_{b_1b_{21}}, p_s, b_{b_{12}b_{211}})\}$*, while their $\mathcal{EL}$, r-graph, as well as UTCQ* `lgg` *is the strictly more general tree-shaped RDF graph:* $\mathcal{G}' = \{(b_{b_1b_2}, p_e, b_{b_{11}b_{21}})\}$*. The latter corresponds to the $\mathcal{G}$ subgraph rooted in $b_{b_1b_2}$ that generalizes the roots b_1 of $\mathcal{G}_1$ and b_2 of $\mathcal{G}_2$.*

8. CONCLUSION AND PERSPECTIVES

We have revisited the Machine Learning problem of computing a *least general generalization (*`lgg`*) of some descriptions* in the setting of RDF and SPARQL. Our contributions significantly extend the state of the art by considering the entire RDF standard and popular conjunctive fragment of SPARQL, i.e., BGPQs. In particular, we neither restrict RDF graphs and BGPQs nor RDF entailment in any way, while closely related works only consider *rooted*-RDF graphs and *unary tree* BGPQs (i.e., describing a *single* root resource or answer variable, respectively), and, further, they *completely ignore* RDF entailment rules (i.e., their particular RDF graphs and BGPQs are simply compared in a *purely relational fashion*). Moreover, in the case of BGPQs, we also endowed with background knowledge the standard RDF entailment relation between queries. As our experiments showed, taking into account the ontological constraints describing the application domain in which queries are posed may enhance the precision of `lgg`s.

As a short-term perspective, we want to study heuristics in order to efficiently prune out as much as possible redundant triples, while computing `lgg`s. Indeed, as for instance Figures 4 and 8 show, our cover graph/query technique produces many redundant triples. This would allow having more readable `lgg`s, as well as reducing the a posteriori elimination effort of redundant triples with standard technique from the literature. Moreover, in the case of BGPQs, removing redundant triples may significantly improves their evaluation time.

Computing `lgg`s has many possible applications in databases (recall Section 1); in particular, we plan to apply our results for BGPQs to the optimization problem of view selection in RDF. Our idea is to select as views to materialize a set $\mathcal{V}$ of `lgg`s of queries from a workload, such that (i) the workload queries can be (partially or totally) rewritten using $\mathcal{V}$ and (ii) $\mathcal{V}$ minimizes a combination of query processing, view storage, and view maintenance costs. This would provide an alternative to [24], which applies to the

so-called RDF database fragment of RDF (restricting RDF entailment) and BGPQs, and which is based on recursively decomposing the workload queries into subqueries that may be materialized.

Finally, we also want to investigate the problem of computing `lgg`s, and apply it to view selection, in the setting of the DL-Lite$_\mathcal{R}$ description logic [15], which underpins OWL2 QL, the other W3C's Semantic Web standard.

Acknowledgment

This work has been partially funded by Lannion-Tregor Communauté and Région Bretagne (PAWS project).

9. REFERENCES

[1] DB2. www.ibm.com/analytics/us/en/technology/db2.
[2] Hadoop. hadoop.apache.org.
[3] Jena. jena.apache.org.
[4] MySQL. www.mysql.com.
[5] Oracle. www.oracle.com/database.
[6] PostgreSQL. www.postgresql.org.
[7] Spark. spark.apache.org.
[8] Virtuoso. virtuoso.openlinksw.com.
[9] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
[10] F. Baader, R. Kiisters, and R. Molitor. Computing least common subsumers in description logics with existential restrictions. In *IJCAI*, 1999.
[11] F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. *Journal of Applied Logic*, 5(3):392 – 420, 2007.
[12] M. A. Bornea, J. Dolby, A. Kementsietsidis, K. Srinivas, P. Dantressangle, O. Udrea, and B. Bhattacharjee. Building an efficient RDF store over a relational database. In *SIGMOD*, 2013.
[13] D. Bursztyn, F. Goasdoué, and I. Manolescu. Optimizing reformulation-based query answering in RDF. In *EDBT*, 2015.
[14] D. Bursztyn, F. Goasdoué, and I. Manolescu. Teaching an RDBMS about ontological constraints. *PVLDB*, 9(12), 2016.
[15] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reasoning*, 39(3), 2007.
[16] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *PODS*, 1998.
[17] M. Chein and M. Mugnier. *Graph-based Knowledge Representation - Computational Foundations of Conceptual Graphs*. Springer, 2009.
[18] W. W. Cohen, A. Borgida, and H. Hirsh. Computing least common subsumers in description logics. In *AAAI*, 1992.
[19] S. Colucci, F. M. Donini, S. Giannini, and E. D. Sciascio. Defining and computing least common subsumers in RDF. *Journal of Web Semantics*, 39:62–80, 2016.
[20] S. Colucci, F. M. Donini, and E. D. Sciascio. Common subsumbers in RDF. In *Advances in Artificial Intelligence (AI*IA)*, pages 348–359, 2013.
[21] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Symposium on Operating System Design and Implementation (OSDI)*, pages 137–150, 2004.
[22] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database systems - the complete book (2. ed.)*. Pearson Education, 2009.
[23] F. Goasdoué, Z. Kaoudi, I. Manolescu, J. Quiané-Ruiz, and S. Zampetakis. Cliquesquare: Flat plans for massively parallel RDF queries. In *ICDE*, pages 771–782, 2015.
[24] F. Goasdoué, K. Karanasos, J. Leblay, and I. Manolescu. View selection in semantic web databases. *PVLDB*, 5(2):97–108, 2011.
[25] F. Goasdoué, I. Manolescu, and A. Roatiş. Efficient query answering against dynamic RDF databases. In *EDBT*, 2013.
[26] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, 3(2-3):158–182, Oct. 2005.
[27] M. F. Husain, J. P. McGlothlin, M. M. Masud, L. R. Khan, and B. M. Thuraisingham. Heuristics-based query processing for large RDF graphs using cloud computing. *IEEE Trans. Knowl. Data Eng.*, 23(9):1312–1327, 2011.
[28] T. Imielinski and W. Lipski. Incomplete information in relational databases. *JACM*, 31(4), 1984.
[29] R. Küsters. *Non-Standard Inferences in Description Logics*, volume 2100 of *LNCS*. Springer, 2001.
[30] K. Lee and L. Liu. Scaling queries over big RDF graphs with semantic hash partitioning. *PVLDB*, 6(14):1894–1905, 2013.
[31] J. Lehmann and L. Bühmann. Autosparql: Let users query your knowledge base. In *ESWC*, 2011.
[32] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. Dbpedia - A large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.
[33] M. Meier. Towards rule-based minimization of RDF graphs under constraints. In *RR*, 2008.
[34] T. Neumann and G. Weikum. x-rdf-3x: Fast querying, high update rates, and consistency for RDF databases. *PVLDB*, 3(1), 2010.
[35] N. Papailiou, D. Tsoumakos, I. Konstantinou, P. Karras, and N. Koziris. H_2rdf+: an efficient data management system for big RDF graphs. In *SIGMOD*, pages 909–912, 2014.
[36] F. Picalausa, Y. Luo, G. H. Fletcher, J. Hidders, and S. Vansummeren. A structural approach to indexing triples. In *ESWC*, 2012.
[37] R. Pichler, A. Polleres, S. Skritek, and S. Woltran. Redundancy elimination on RDF graphs in the presence of rules, constraints, and queries. In *Web Reasoning and Rule Systems*, 2010.
[38] R. Pichler, A. Polleres, S. Skritek, and S. Woltran. Complexity of redundancy detection on RDF graphs in the presence of rules, constraints, and queries. *Semantic Web*, 4(4):351–393, 2013.
[39] G. D. Plotkin. A note on inductive generalization.

Machine Intelligence, 5, 1970.

[40] G. D. Plotkin. A further note on inductive generalization. *Machine Intelligence*, 6, 1971.

[41] R. Ramakrishnan and J. Gehrke. *Database management systems (3. ed.)*. McGraw-Hill, 2003.

[42] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, Jan. 1965.

[43] J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning*. Elsevier and MIT Press, 2001.

[44] J. Urbani, S. Kotoulas, J. Maassen, F. van Harmelen, and H. E. Bal. Webpie: A web-scale parallel inference engine using mapreduce. *J. Web Sem.*, 10:59–75, 2012.

[45] J. Urbani, S. Kotoulas, E. Oren, and F. van Harmelen. Scalable distributed reasoning using mapreduce. In *ISWC*, pages 634–649, 2009.

[46] Resource Description Framework 1.1. https://www.w3.org/TR/rdf11-concepts.

[47] RDF 1.1 Semantics. https://www.w3.org/TR/rdf11-mt/.

[48] SPARQL 1.1. https://www.w3.org/TR/sparql11-query/.

[49] C. Weiss, P. Karras, and A. Bernstein. Hexastore: sextuple indexing for semantic web data management. *PVLDB*, 1(1):1008–1019, 2008.

[50] B. Zarrieß and A. Turhan. Most specific generalizations w.r.t. general EL-TBoxes. In *IJCAI*, 2013.

APPENDIX

A. LUBM QUERIES

$Q1(?X, ?Y) : -$ $?X$ "$http://www.w3.org/1999/02/22\text{-}rdf\text{-}syntax\text{-}ns\#type$" "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#Employee$", $?X$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#worksFor$" "$http://www.Department0.University0.edu$", $?X$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#degreeFrom$" $?Y$
$Q2(?X, ?Y) : -$ $?X$ "$http://www.w3.org/1999/02/22\text{-}rdf\text{-}syntax\text{-}ns\#type$" "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#Employee$", $?X$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#worksFor$" "$http://www.Department0.University0.edu$", $?X$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#doctoralDegreeFrom$" $?Y$
$Q3(?X, ?Y, ?Z) : -$ $?X$ "$http://www.w3.org/1999/02/22\text{-}rdf\text{-}syntax\text{-}ns\#type$" "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#Student$", $?X$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#advisor$" $?Y$, $?Y$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#teacherOf$" $?Z$, $?X$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#takesCourse$" $?Z$
$Q4(?X, ?Y) : -$ $?X$ "$http://www.w3.org/1999/02/22\text{-}rdf\text{-}syntax\text{-}ns\#type$" "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#Faculty$", $?X$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#degreeFrom$" $?Y$, $?X$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#memberOf$" $?Y$
$Q5(?X, ?Y) : -$ $?X$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#degreeFrom$" $?Y$, $?X$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#memberOf$" "$http://www.Department0.University0.edu$",
$Q6(?W, ?X, ?Y) : -$ $?X$ "$http://www.w3.org/1999/02/22\text{-}rdf\text{-}syntax\text{-}ns\#type$" "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#GraduateStudent$", $?Y$ "$http://www.w3.org/1999/02/22\text{-}rdf\text{-}syntax\text{-}ns\#type$" "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#Faculty$", $?W$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#publicationAuthor$" $?X$, $?W$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#publicationAuthor$" $?Y$
$Q7(?W, ?X, ?Y) : -$ $?X$ "$http://www.w3.org/1999/02/22\text{-}rdf\text{-}syntax\text{-}ns\#type$" "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#GraduateStudent$", $?X$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#advisor$" $?Y$, $?Y$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#teacherOf$" $?Z$, $?X$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#takesCourse$" $?Z$, $?W$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#publicationAuthor$" $?X$, $?W$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#publicationAuthor$" $?Y$
$Q8(?X, ?Y) : -$ $?X$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#doctoralDegreeFrom$" $?Z$, $?X$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#teacherOf$" $?Y$
$Q9(?X, ?Y) : -$ $?X$ "$http://www.w3.org/1999/02/22\text{-}rdf\text{-}syntax\text{-}ns\#type$" "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#Faculty$", $?X$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#degreeFrom$" "$http://www.University532.edu$", $?X$ "$http://swat.cse.lehigh.edu/onto/univ\text{-}bench.owl\#memberOf$" $?Y$

Figure 9: LUBM queries

B. DBPEDIA QUERIES

$Q1(?X, ?Y) :-$ $?X$ "$http://www.w3.org/1999/02/22\text{-}rdf\text{-}syntax\text{-}ns\#type$" "$http://dbpedia.org/ontology/Organisation$", $?Y$ "$http://www.w3.org/1999/02/22\text{-}rdf\text{-}syntax\text{-}ns\#type$" "$http://dbpedia.org/ontology/Organisation$", $?X$ "$http://dbpedia.org/ontology/locationCountry$" "$http://dbpedia.org/resource/France$", $?X$ "$http://dbpedia.org/ontology/parentCompany$" $?Y$
$Q2(?X, ?Y) :-$ $?X$ "$http://www.w3.org/1999/02/22\text{-}rdf\text{-}syntax\text{-}ns\#type$" "$http://dbpedia.org/ontology/Airport$", $?X$ "$http://www.ontologydesignpatterns.org/ont/dul/DUL.owl\#hasLocation$" "$http://dbpedia.org/resource/France$", $?X$ "$http://dbpedia.org/ontology/city$" $?Z$, $?X$ "$http://dbpedia.org/ontology/owner$" $?Y$, $?Y$ "$http://www.w3.org/1999/02/22\text{-}rdf\text{-}syntax\text{-}ns\#type$" "$http://dbpedia.org/ontology/Person$", $?V$ "$http://www.ontologydesignpatterns.org/ont/dul/DUL.owl\#hasLocation$" $?X$
$Q3(?X, ?Y) :-$ $?X$ "$http://www.w3.org/1999/02/22\text{-}rdf\text{-}syntax\text{-}ns\#type$" "$http://dbpedia.org/ontology/Single$", $?X$ "$http://dbpedia.org/ontology/musicalArtist$" $?Y$, $?Y$ "$http://www.ontologydesignpatterns.org/ont/dul/DUL.owl\#hasLocation$" $?Z$, $?Y$ "$http://dbpedia.org/ontology/genre$" $?V$,
$Q4(?X, ?Y) :-$ $?X$ "$http://dbpedia.org/ontology/birthPlace$" $?V$, $?X$ "$http://dbpedia.org/ontology/parent$" $?Y$, $?Y$ "$http://dbpedia.org/ontology/deathPlace$" $?V$, $?Z$ "$http://www.w3.org/1999/02/22\text{-}rdf\text{-}syntax\text{-}ns\#type$" "$http://dbpedia.org/ontology/Film$", $?Z$ "$http://dbpedia.org/ontology/starring$" $?X$, $?Z$ "$http://dbpedia.org/ontology/director$" $?Y$
$Q5(?X, ?Y) :-$ $?X$ "$http://dbpedia.org/ontology/deathPlace$" "$http://dbpedia.org/resource/France$", $?X$ "$http://dbpedia.org/ontology/parent$" $?Y$, $?Z$ "$http://www.w3.org/1999/02/22\text{-}rdf\text{-}syntax\text{-}ns\#type$" "$http://dbpedia.org/ontology/Film$", $?Z$ $?V$ $?Y$
$Q6(?X, ?Y) :-$ $?X$ "$http://dbpedia.org/ontology/birthPlace$" "$http://dbpedia.org/resource/France$", $?Y$ "$http://dbpedia.org/ontology/birthPlace$" "$http://dbpedia.org/resource/France$", $?X$ "$http://dbpedia.org/resource/predecessor$" $?Y$, $?Z$ "$http://www.w3.org/1999/02/22\text{-}rdf\text{-}syntax\text{-}ns\#type$" "$http://dbpedia.org/ontology/Cartoon$", $?Z$ "$http://dbpedia.org/ontology/animator$" $?X$, $?Z$ "$http://dbpedia.org/ontology/director$" $?Y$
$Q7(?X, ?Y) :-$ $?X$ "$http://dbpedia.org/ontology/birthPlace$" "$http://dbpedia.org/resource/France$", $?Y$ "$http://dbpedia.org/ontology/birthPlace$" "$http://dbpedia.org/resource/France$", $?X$ "$http://dbpedia.org/resource/parent$" $?Y$, $?Z$ "$http://www.w3.org/1999/02/22\text{-}rdf\text{-}syntax\text{-}ns\#type$" "$http://dbpedia.org/ontology/Film$", $?Z$ "$http://dbpedia.org/ontology/starring$" $?X$, $?Z$ "$http://dbpedia.org/ontology/director$" $?Y$
$Q8(?X, ?Y) :-$ $?X$ "$http://dbpedia.org/ontology/deathPlace$" "$http://dbpedia.org/resource/England$", $?Y$ "$http://dbpedia.org/ontology/deathPlace$" "$http://dbpedia.org/resource/England$", $?X$ "$http://dbpedia.org/resource/parent$" $?Y$, $?Z$ "$http://www.w3.org/1999/02/22\text{-}rdf\text{-}syntax\text{-}ns\#type$" "$http://dbpedia.org/ontology/Film$", $?Z$ "$http://dbpedia.org/ontology/starring$" $?X$, $?Z$ "$http://dbpedia.org/ontology/director$" $?Y$

Figure 10: DBPEDIA queries