Ontology-Based RDF Integration of Heterogeneous Data

Maxime Buron1 François Goasdoué2 Ioana Manolescu1 Marie-Laure Mugnier3
1Inria and LIX (UMR 7161, CNRS and Ecole polytechnique), France 2Univ. Rennes, CNRS, IRISA, France 3Univ. Montpellier, LIRMM, Inria, France

ABSTRACT

The proliferation of heterogeneous data sources in many application contexts brings an urgent need for expressive and efficient data integration mechanisms. There are strong advantages to using RDF graphs as the integration format: being schemaless, they allow for flexible integration of data from heterogeneous sources; RDF graphs can be interpreted with the help of an ontology, describing application semantics; last but not least, RDF enables joint querying of the data and the ontology.

To address this need, we formalize RDF Integration Systems (RIS), Ontology-Based-Data Access mediators, that go beyond the state of the art in the ability to expose, integrate and flexibly query data from heterogeneous sources through GLAV (global-local-as-view) mappings. We devise several query answering strategies, based on an innovative integration of LAV view-based rewriting and a form of mapping saturation. Our experiments show that one of these strategies brings strong performance advantages, resulting from a balanced use of mapping saturation and query reformulation.

1 INTRODUCTION

The proliferation of digital data sources across all application domains brings a new urgency to the need for tools which allow to query heterogeneous data (relational, JSON, key-values, graphs etc.) in a flexible fashion. Traditional data integration systems fall into two classes: data warehousing, where all data source content is materialized in a single centralized source, respectively, mediation, where data remains in its original stores and all data can be queried through a single module called mediator. Data warehousing simplifies query evaluation, but requires potentially costly maintenance operations when the content of data sources changes; mediation does not suffer from these drawbacks, but requires more intricate query evaluation algorithms to distribute the work between the sources and the mediator.

Below, we classify prior mediator-based approaches according to two main dimensions, and illustrate this classification in Table 1. Note that we also include in this table theoretical frameworks that did not necessarily lead to implementations.

A first dimension concerns the data model and query language provided by the mediator to its applications.

(i) The earliest goal of a mediator system was to mimic a single, integrated database. Thus the mediator supports one data model and its query language, e.g., relational and SQL, or XML and XPath/XQuery. More recent polystore systems support side-by-side different (data model, query language) pairs. These database-style mediators appear in the row we label DB in Table 1.

(ii) Mediators studied in knowledge representation and management research provide a view of the data sources as a set of classes and relationships, also endowed with a set of semantic constraints, or ontology. In such systems, users formulate conjunctive (relational) queries; answering them involves not only evaluation over the data (as done in DB mediators), but also reasoning on the data with the help of ontologies. This mediation approach is also commonly termed Ontology-Based Data Access (OBDA) [41], with ontologies expressed in Description Logics (DL, in short). Work following this approach are listed in the row we label CQ in Table 1.

(iii) RDF [47] is naturally suited as an integration model, thanks to its flexibility, its wide adoption in the Open Data community, its close relationship with ontology languages such as RDFS and OWL, and the presence of its associated standard SPARQL query language. Accordingly, several mediators from the CQ group have been extended to support RDF as an integration model and SPARQL query answering. However, while SPARQL allows querying the data together with the ontology, e.g., “find the properties of node $n$, as well the classes to which the values of these properties belong”, a DL-based mediation approach shares with all logic-based query languages, e.g., Datalog, SQL etc., the inability to do so. RDF mediators which support SPARQL but limited to querying the data only (not the ontology) appear in the row we label SPARQL-data in Table 1.

(iv) Recent RDF mediators lift this limitation to support joint querying of the data and ontology; we list them in the SPARQL row in Table 1.

A second dimension is how the source (or local) schemas are connected to the global (integration) schema, using mappings [23]. There are three types of mappings, each corresponding to a column in Table 1. The simplest mappings define each element of the global schema, e.g., each relation (if the global schema is relational), as a view over the local schemas; this is known as Global-As-View, or GAV in short. In a GAV system, a query over the global (virtual) schema is easily transformed into a query over the local schemas, by unfolding each global schema relation, i.e., replacing it with its definition. In contrast, Local-As-View (LAV) mappings define elements of the local schemas as views over the global one. Query answering in this context requires rewriting the query with the views describing the local sources [31]. Global-Local-As-View (GLAV) data integration generalizes both GAV and LAV. A GLAV mapping pairs a query $q_1$ over one or several local schemas to a query $q_2$ over the global schema, having the same answer variables. The semantics is that, for each answer of $q_1$, the integration system exposes the data comprised in a corresponding answer of $q_2$. GLAV maximizes flexibility, or, equivalently, integration expressive power: unlike LAV, a GLAV mapping may expose only part of a given source’s data, and may combine data from several sources; unlike GAV, a GLAV mapping may include joins or complex expressions over

Table 1: Outline of the positioning of our work.

<table>
<thead>
<tr>
<th>Model</th>
<th>Mappings</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td>[22, 24, 27]</td>
</tr>
<tr>
<td>CQ</td>
<td>[4, 5, 22, 38]</td>
</tr>
<tr>
<td>SPARQL-data</td>
<td>[1, 28, 30, 36]</td>
</tr>
<tr>
<td>SPARQL</td>
<td>[19]</td>
</tr>
</tbody>
</table>

Table 1: Outline of the positioning of our work.
the global schema.

In this work, we study GLAV mediation supporting SPARQL queries over the data and the ontology. We pick GLAV for its highest expressive power, RDF for its wide adoption, and aim at querying the data and the ontology in order to fully benefit from the flexibility and expressivity of RDF. As Table 1 shows, our system is the first capable of integrating multiple data sources through GLAV mappings, for SPARQL querying over the data and the ontology; further, it supports heterogeneous data sources (of different data models). A benefit of our using GLAV is the ability to support a form of incomplete information, naturally present in RDF through the so-called blank nodes, in the virtual RDF graph exposed by the mediator (see Section 3.1).

Our closest competitors only support GAV mappings, even though some support more expressive ontologies and/or queries [16, 33, 44]. Formal OBDA frameworks based on GLAV mappings have been defined, e.g., [18], without concretely deployed systems. A technique for simulating GLAV mappings through GAV ones under certain conditions is suggested in [21], however this solution has many drawbacks; we defer a detailed discussion to Section 6.

Contributions and novelty The contributions we make in this work are as follows.

1. RIS Formalism We formally define RDF Integration Systems (RIS, in short). OBDA mediators capable of exposing data from heterogeneous sources of virtually any data model through GLAV mappings, under the form of an RDF graph endowed with an RDFS ontology. We formalize the problem of BGP (basic graph pattern) RDF query answering over the RDF data and ontology exposed in such systems.

2. Novel RIS query answering techniques We describe several RIS query answering methods based on transforming mappings into LAV view definitions, and on reducing query answering to rewriting it using views. Our first method combines known techniques; the other two methods are novel, and rely on a form of mapping saturation. We show that a smart decomposition of reasoning between offline precomputation and query time makes one of these methods much faster than the others.

The paper is organized as follows. Section 2 recalls a set of preliminary notions we build upon. Then, Section 3 defines our RIS and formalizes RIS query answering. Section 4 describes RIS query answering methods. Section 5 presents our experiments, then we discuss related work and conclude.

2 PRELIMINARIES

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

Then, we recall two techniques, namely query reformulation (Section 2.4) and view-based query rewriting (Section 2.5), which will serve as building blocks for our query answering techniques.

2.1 RDF Graphs

We consider three pairwise disjoint sets of values: \( \mathcal{F} \) of IRIs (resource identifiers), \( \mathcal{L} \) of literals (constants) and \( \mathcal{B} \) of blank nodes modeling unknown IRIs or literals, a.k.a. labelled nulls [3, 29]. A well-formed triple belongs to \( \mathcal{F} \cup \mathcal{B} \times \mathcal{L} \cup \mathcal{L} \cup \mathcal{F} \), and an RDF graph \( G \) is a set of (well-formed) triples. A triple \((s, p, o)\) states that its subject \( s \) has the property \( p \) with the object value \( o \) [47]. We denote by \( \text{Val}(G) \) the set of all values (IRIs, blank nodes and literals) occurring in an RDF graph \( G \), and by \( \text{Bl}(G) \) its

<table>
<thead>
<tr>
<th>Schema triples</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subclass</td>
<td>( (s, \lessdot_s, o) )</td>
</tr>
<tr>
<td>Subproperty</td>
<td>( (s, \lessdot_{sp}, o) )</td>
</tr>
<tr>
<td>Domain typing</td>
<td>( (s, \leadsto, o) )</td>
</tr>
<tr>
<td>Range typing</td>
<td>( (s, \leadsto_r, o) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data triples</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class fact</td>
<td>( (s, r, o) )</td>
</tr>
<tr>
<td>Property fact</td>
<td>( (s, p, o) \text{ s.t. } p \notin {r, \lessdot_s, \lessdot_{sp}, \leadsto, \leadsto_r} )</td>
</tr>
</tbody>
</table>

Table 2: RDF triples.

set of blank nodes. In triples, we use \_b (possibly with indices) to denote blank nodes, and quoted strings to denote literals. Within an RDF graph, we distinguish data triples from schema ones. The former describe data (either attach a type, or a class, to a resource, or state the value of a certain data property of a resource). The latter state ontological constraints using RDF Schema (RDFS), which relate classes and properties: subclass (specialization relation between types), subproperty (specialization of a binary relation), typing of the domain (first attribute) of a property, respectively, range (typing of the second attribute) of a property. Table 2 introduces short notations we adopt for these schema properties.

From now on, we denote by \( \mathcal{F}_{\text{res}} \) the reserved IRIs from the RDF standard, e.g., the properties \( \lessdot_s, \lessdot_{sp}, \leadsto, \leadsto_r \), shown in Table 2. The rest of the IRIs are application-dependent classes and properties, which are said user-defined and denoted by \( \mathcal{F}_{\text{user}} \).

Hence, \( \mathcal{F}_{\text{user}} = \mathcal{F} \setminus \mathcal{F}_{\text{res}} \).

We will consider RDF graphs with RDFS ontologies made of schema triples of the four above flavours. More precisely:

Definition 2.1 (RDFS ontology) An ontology triple is a schema triple whose subject and object are user-defined IRIs from \( \mathcal{F}_{\text{user}} \). An RDFS ontology (or ontology in short) is a set of ontology triples. Ontology \( O \) is the ontology of an RDF graph \( G \) if \( O \) is the set of schema triples of \( G \).

Above, ontology triples are not allowed over blank nodes. This is to only simplify the presentation; we could have allowed them, and handled them as in [29]. More importantly, we forbid ontology triples from altering the common semantics of RDF itself. For instance, we do not allow \( \lessdot_{\text{sp}}, \leadsto_{\text{sp}}, \leadsto_{\text{r}} \), which would impose that the range of every property shares all the types of the property’s domain! This second restriction can be seen as common-sense; it underlies most ontological formalisms, in particular description logics [8] thus the W3C’s Web Ontology Language (OWL), Datalog++ [15] and existential rules [39], etc.

Example 2.2 (Running example, based on [12]). Consider the following RDF graph:

\[
G_{ex} = \{\langle \text{worksFor}, \leadsto_{\text{sp}}, \text{Person} \rangle, \langle \text{worksFor}, \leadsto_{\text{r}}, \text{Org} \rangle, \\
\langle \text{PubAdmin}, \lessdot_{\text{sp}}, \text{Org} \rangle, \langle \text{Comp}, \lessdot_{\text{sp}}, \text{Org} \rangle, \\
\langle \text{NatComp}, \lessdot_{\text{sp}}, \text{Comp} \rangle, \langle \text{hiredBy}, \lessdot_{\text{sp}}, \text{worksFor} \rangle, \\
\langle \text{CEOOf}, \lessdot_{\text{sp}}, \text{worksFor} \rangle, \langle \text{CEOOf}, \leadsto_{\text{r}}, \text{Comp} \rangle, \\
\langle \text{p1}, \text{CEOOf, } \_b c \rangle, \langle \_b c, \_b r, \text{NatComp} \rangle, \\
\langle \text{p2}, \text{hiredBy, } \_a, \_a, \text{PubAdmin} \rangle\}
\]

The ontology of \( G_{ex} \), i.e., the first eight schema triples, states that people work for organizations, some of which are public administrations or companies. Further, national companies are a kind of companies. Being hired by or being CEO of an organization are two ways of working for it; in the latter case, this organization is a company. The facts of \( G_{ex} \), i.e., the four remaining data triples, state that \( p1 \) is CEO of some unknown company represented by the blank node \( \_b c \), which is a national company, and \( p2 \) is hired by the public administration \( \_a \).
2.2 RDF Entailment Rules

An entailment rule $r$ has the form $\text{body}(r) \rightarrow \text{head}(r)$, where $\text{body}(r)$ and $\text{head}(r)$ are RDF graphs, respectively called body and head of the rule $r$. In this work, we consider the RDFS entailment rules $R$ shown in Table 3, which are the most frequently used; in the table, all values except RDF reserved IRIs are blank nodes. For instance, rule rdfs5 reads: whenever in an RDF graph, a property $p_1$ is a subproperty of a property $p_2$, and further $p_2$ is a subproperty of $p_3$ (body of rdfs5), it follows that $p_1$ is a subproperty of $p_2$ and a resource $s$ has the value $o$ for $p_1$, then $s$ also has $o$ as a value for $p_2$. The triples $(p_1, \vartriangleleftSP, p_3)$ and $(s, p_2, o)$ in the above examples are called implicit, i.e., they hold in a graph thanks to the entailment rules, even if they may not be explicitly present in the graph. Following [12], we view $R$ as partitioned into two subsets: the rules $R_c$ lead to implicit schema triples, while rules $R_a$ lead to implicit data triples. The direct entailment of an RDF graph $G$ with a set of RDF entailment rules $R$, denoted by $G \models_{R_c, R_a} G$, is the set of implicit triples resulting from rule applications that use solely the explicit triples of $G$. For instance, the rule rdfs9 is applied to the graph $G_{\text{ex}}$, which comprises $(\text{NatComp}, \vartriangleleft_{\text{SC}}, \text{Comp}), (\_b_3, r, \text{NatComp})$, leading to the implicit triple $(\_b_3, r, \text{Comp})$. This triple belongs to $C_{G_{\text{ex}}, R}$ (hence also to $C_{G_{\text{ex}}, R}$). The saturation of an RDF graph allows materializing its semantics, by iteratively augmenting it with the triples it entails using entailment rules, until reaching a fixpoint; this process is finite [48]. Formally:

**Definition 2.3 (RDF graph saturation).** Let $G$ be an RDF graph and $R$ a set of entailment rules. We recursively define a sequence $(G_i)_{i \geq 0}$ of RDF graphs as follows: $G_0 = G$, and $G_{i+1} = G_{i} \cup C_{G_{i}} R$ for $i \geq 0$. The saturation of $G$ w.r.t. $R$, denoted by $G^R$, is $G_n$ for $n$ the smallest integer such that $G_n = G_{n+1}$.

**Example 2.4 (Saturation).** The saturation of $G_{\text{ex}}$ w.r.t. the set $R$ of RDFS entailment rules shown in Table 3 is attained after the following two saturation steps:

$$G_{\text{ex}} := G_{\text{ex}} \cup \{(\text{NatComp}, \vartriangleleft_{\text{SC}}, \text{Org}), (\text{hiredBy}, \vartriangleleft_{=}, \text{Person}), (\text{hiredBy}, \vartriangleright_{\text{r}}, \text{Org}), (\text{CEOOf}, \vartriangleleft_{=}, \text{Person}), (\text{CEOOf}, \vartriangleright_{\text{r}}, \text{Org}), (p_1, \text{worksFor}, \_b_3), (_b_3, r, \text{Comp}), (p_2, \text{worksFor}, a), (\text{a, r}, \text{Org})\}$$

2.3 Basic Graph Pattern Queries

A popular RDF query dialect consists of basic graph pattern queries, or BGPQs, in short. Let $\gamma$ be a set of variable symbols, disjoint from $\cup R \cup \forall \gamma$. A basic graph pattern (BGP) is a set of triple patterns (triples in short) belonging to $(\exists \cup \forall \gamma \cup \forall \gamma) \times (\exists \gamma \cup \forall \gamma \cup \forall \gamma)$. For a BGP $P$, we denote by $\text{Var}(P)$ the set of variables occurring in $P$, by $\text{Bl}(P)$ its set of blank nodes, and by $\text{Val}(P)$ its set of values (IRIs, blank nodes, literals and variables).

**Definition 2.5 (BGP Queries).** A BGP query $q$ is of the form $q(x) \leftarrow P$, where $P$ is a BGP (also denoted by $\text{body}(q)$), and $x \subseteq \text{Var}(P)$ are the answer variables of $q$.

To ease the presentation, and without loss of generality, we consider BGPQs without blank nodes, as it is well-known that these can be replaced by non-answer variables [46].

For query answering based on query reformulation (see Section 2.4), it is convenient to slightly generalize BGPQs into partially instantiated BGPQs [12, 29]. Starting from a BGPQ $q$, partial instantiation may replace some variables with values from $\exists \cup \forall \gamma \cup \forall \gamma$, as specified by a substitution $\sigma$. To $\sigma$, and in contrast with standard BGPQs, some answer variables of the resulting query $q_\sigma$ can be bound:

**Example 2.6 (Partially instantiated BGPQ).** Consider the BGPQ asking for who is working for which kind of company $q(x,y) \leftarrow (x, \text{worksFor}, z), (z, r, y), (y, \vartriangleleft_{\text{SC}}, \text{Comp})$ and the substitution $\sigma = \{ x \mapsto p_1 \}$. The corresponding partially instantiated BGPQ is $q(p_1, y) \leftarrow (p_1, \text{worksFor}, z), (z, r, y), (y, \vartriangleleft_{\text{SC}}, \text{Comp})$. In it, the first answer variable has been bound to $p_1$.

For simplicity, below we use the term "query" to designate either a standard BGPQ or a partially instantiated BGPQ.

The semantics of a BGPQ on an RDF graph is defined through standard homomorphisms from the query body to the queried graph. We recall that a homomorphism from a BGP $P$ to an RDF graph $G$ is a function $\psi$ from $\text{Val}(P)$ to $\text{Val}(G)$ such that for any triple $(s, p, o) \in P$, the triple $(\psi(s), \psi(p), \psi(o))$ is in $G$, with $\psi$ the identity on IRIs and literals. Next, we distinguish query evaluation, whose result is just based on the explicit triples of the graph, i.e., on BGP-to-RDF graph homomorphisms, from query answering that also accounts for the implicit graph triples, resulting from entailment. Formally:

**Definition 2.7 (Evaluation and answering).** The answer set to a BGPQ $q$ on an RDF graph $G$ w.r.t. a set $R$ of RDF entailment rules is:

$$q(G, R) = \{ \phi(x) \mid \phi \text{ homomorphism from } \text{body}(q) \text{ to } G^R \}.$$

If $x = \emptyset$, $q$ is a Boolean query, in which case $q$ is false when $q(G, R) = \emptyset$ and true when $q(G, R) = \{(\downarrow)\}$, i.e., the answer to $q$ is the empty tuple.

The evaluation of $q$ on $G$, denoted $q(G, \emptyset)$ or $q(G)$ in short, is obtained from homomorphisms from $\text{body}(q)$ to $G$ alone (not $G^R$). It can be seen as a particular case of query answering when $R = \emptyset$.

**Example 2.8 (Evaluation and answering).** Consider again the BGPQ $q$ from the preceding example. Its evaluation on $G_{\text{ex}}$ is empty because $G_{\text{ex}}$ has no explicit worksFor assertion, while its answer set on $G_{\text{ex}}$ w.r.t. $R$ is $\{(p_1, \text{NatComp})\}$ because $p_1$ being CEO of $\_b_3$ implicitly works for it, and $\_b_3$ is explicitly a company of the particular type $\text{NatComp}$.

The above notions and notations naturally extend to unions of (partially instantiated) BGPQs, or UBGPQs in short.

We end this section by pointing out that many RDF data management systems use saturation-based query answering, which directly follows the definition of query answering: an RDF graph $G$ is first saturated with the set $R$ of entailment rules, so that the
answer set to an incoming query \( q \) is obtained through query evaluation as \( q(G^R) \).

2.4 Query Reformulation

Reformulation-based query answering is an alternative technique to the widely adopted saturation-based query answering. It consists in reformulating a query using \( \mathcal{R} \), so that evaluating the reformulated query on \( G \) yields the answer set to the original query on \( G \) w.r.t. \( \mathcal{R} \). Intuitively, reformulation injects the ontological knowledge into the query, just as saturation injects it into the RDF graph. We rely here on the very recent algorithm from [12], which takes into account all the entailment rules from Table 3. The process is decomposed into two steps according to the partition of \( \mathcal{R} \) into \( \mathcal{R}_a \) and \( \mathcal{R}_c \).

(i) The first step reformulates a BGPQ \( q \) w.r.t. an RDFS ontology \( O \) and the set of rules \( \mathcal{R}_c \) into a UBGPQ, say \( Q_c \), which is guaranteed not to contain ontological triples. Intuitively, this step generates new BGPQs obtained from \( q \) by instantiating variables that query the ontology with all their possible bindings; for instance, \( y \) in a query triple \((y, \prec_c, {\text{Comp}})\) is bound to the IRIs of all explicit and implicit subclasses of \( \text{Comp} \) in \( O \). This step, alone, is sound and complete w.r.t. \( \mathcal{R}_c \) for query answering, i.e., for any graph \( G \) with ontology \( O \), \( q(G, \mathcal{R}_a) = Q_c(G) \).

(ii) The second step reformulates \( Q_c \) w.r.t. \( O \) and \( \mathcal{R}_a \) and outputs a UBGPQ, say \( Q_{c,a} \). This step, alone, is sound and complete w.r.t. \( \mathcal{R}_a \) for query answering, i.e., for any graph \( G \) with ontology \( O \), \( Q_c(G, \mathcal{R}_a) = Q_{c,a}(G) \). Furthermore, a key property is that \( q(G, \mathcal{R}) = Q_{c,a}(G) \), i.e., only \( \mathcal{R}_a \) needs to be considered to answer \( Q_c \) with respect to the entire set of rules \( \mathcal{R} \). This is the fundamental reason why the successive application of these two reformulation steps leads to a sound and complete reformulation-based query answering technique: \( q(G, \mathcal{R}) = Q_{c,a}(G) \).

Example 2.9 (Two-step reformulation). Consider the query \( q(x, y) \leftarrow (x, \text{worksFor}, z), (z, \tau, y), (y, \prec_c, {\text{Comp}}) \) from the preceding example and the ontology \( O \) in Example 2.2. The first reformulation step instantiates the triple \((y, \prec_c, \text{Comp})\) on \( O \), leading to: \( Q_c = q(x, \text{NatComp}) \leftarrow (x, \text{worksFor}, z), (z, \tau, \text{NatComp}) \). Then, \( Q_c \) is reformulated into \( Q_{c,a} = q(x, \text{NatComp}) \leftarrow (x, \text{worksFor}, z), (z, \tau, \text{NatComp}) \cup q(x, \text{NatComp}) \leftarrow (x, \text{hireFor}, z), (z, \tau, \text{NatComp}) \cup q(x, \text{NatComp}) \leftarrow (x, \text{ceoOf}, z), (z, \tau, \text{NatComp}) \) by specializing \text{worksFor} according to its subproperties in \( O \). It can be checked that \( Q_{c,a}(G_{ex}) = q(G_{ex}, \mathcal{R}) = q(G_{ex}^R) = \{(p_1, \text{NatComp})\} \), obtained here from the third BGPQ in \( Q_{c,a} \).

2.5 Query Rewriting-based Data Integration

We recall now the basics of relational view-based query rewriting (Section 2.5.1), which has been extensively studied [23, 31]. Then we present a generalization of the notion of views as mappings [35] (Section 2.5.2).

2.5.1 View-based (LAV) Data Integration. An integration system \( I \) is made of a global schema \( S \) (a set of relations) and a set \( V \) of views. An instance of \( I \) assigns a set of tuples to each relation of \( S \) and to each view of \( V \). The data stored in a view is called its extension. Further, to each view \( V \) is associated a query \( V(q) \sim \psi(q) \) over the global schema \( S \), specifying how its data fits into \( S \). Accordingly, this framework is called local-as-view (LAV) data integration. For instance, let \( S \) consist of three relations \( \text{Emp}(\text{id}, \text{name}, \text{did}), \text{Dept}(\text{id}, \text{did}, \text{country}), \text{Salary} (\text{id}, \text{amount}) \), where \( \text{id}, \text{did} \) and \( \text{did} \) are respectively identifiers for employees, departments and companies.

Consider the views \( V_1(\text{id}, \text{name}, \text{country}) \sim \text{Emp}(\text{id}, \text{name}, \text{did}), \text{Dept}(\text{id}, \text{did}, \text{country}), \text{Salary}(\text{id}, \text{amount}) \), \( V_2(\text{id}, \text{country}) \sim \text{Emp}(\text{id}, \text{name}, \text{did}), \text{Dept}(\text{id}, \text{did}, \text{country}), \text{Salary}(\text{id}, \text{amount}) \), providing the names of IBM employees and where they work, and \( V_2(\text{id}, \text{amount}) \sim \text{Emp}(\text{id}, \text{name}, \text{did}), \text{Dept}(\text{id}, \text{did}, \text{country}), \text{Salary}(\text{id}, \text{amount}) \), which indicates the salaries of employees in R&D departments. Typically, no single view is expected to bring all information of a given kind; for instance, \( V_1 \) brings some IBM employees, but other views may bring others, e.g., \( V_2 \), possibly overlapping with \( V_1 \); this is called the “Open World Assumption” (OWA).

In an OWA setting, we are interested in certain answers [31], i.e., those that are sure to be part of the query result, knowing the data present in the views. Such answers can be computed by rewriting a query over \( S \), into one over the views \( V \), evaluating the rewriting over the view extensions produces the answers.

Ideally, a rewriting should be equivalent to the query over \( S \), i.e., compute exactly the same answers. However, depending on the views and queries, such a rewriting may not exist. For instance, the query \( q(n, a) : \text{Emp}(n, d), \text{Dept}(d, \text{"France"}) \), \( \text{Salary}(e, a) \) does not have an equivalent rewriting using \( V_1 \) and \( V_2 \), because \( V_1 \) only provides IBM employees working in France, while \( V_2 \) only has salaries of employees of R&D departments. Maximally contained rewriting brings all the query answers that can be obtained through the given set of views; the rewriting may be not equivalent to \( q \) (but just contained in \( q \)). In our example, \( q(n, a) : V_1(n, d, \text{"France"}), V_2(e, a) \) is a maximally contained rewriting of \( q \); it returns employees of French IBM R&D departments with their salary, clearly a subset of \( q \) answers.

A remarkable result holds for (unions of) conjunctive queries (UCQs), conjunctive views (views \( V \) such that the associated query \( V(q) \sim \psi(q) \) is a CQ) and rewritings that are UCQs: any maximally contained rewriting computes exactly the certain answers [2]; we will build upon this result for answering queries in our RDF integration systems.

2.5.2 GLAV Data Integration. The above setting has been generalized to views that are not necessarily stored as such, but just queries over some underlying data source. For instance, assuming a data source \( D \) holds the relations \( \text{Person}(\text{id}, \text{name}) \) and \( \text{Contract}(\text{id}, \text{did}, \text{country}) \) (see Figure 1) with people and their work contracts at IBM, the view \( V_1 \) from the above example may be defined on \( D \) by the query \( V_1^D \) over the \( D \) schema shown in the figure (note that \( V_1^D \) hides the department from system \( I \) ); \( V_1^D \) provides the extension of \( V_1 \). Similarly, view \( V_2 \) may be defined as a query over some data source (or sources).

![Figure 1: Example: view \( V_1 \) as a GLAV mapping.](image-url)
Historically, two restrictions of GLAV mappings have been investigated. First, global-as-view (or GAV) mappings define global schema relations as views over the local schemas. Specifically, a GAV mapping \( q_1(x) \leadsto q_2(x) \), \( q_2 \) defines a single element of the global schema (hence body \( q_2 \)) is restricted to a single atom if \( q_2 \) is a CQ, or a single triple pattern if \( q_2 \) is a BGPQ) and its variables are exactly \( x \). Second, local-as-view (LAV) mappings express elements of the local schema as views over the global schema, similarly to the views described in Section 2.5.1. Importantly, unlike GAV mappings, GLAV ones do not require all variables of \( q_2 \) to be answer variables (e.g., \( dID \), \( v_1 \) in Figure 1); this makes integration more powerful. For example, suppose that (1, “John Doe”, “France”) is an answer to \( V^1_1 \) above. Then, \( V_1 \) exposes this tuple in the global schema as: \( \text{Emp}(1, \text{"John Doe"}, \text{"France"}), \text{Dept}(x, \text{"IBM"}, \text{"France"}) \), stating that John Doe works for a department whose location is in France. Here, \( x \) is an existential variable (called "labeled null" in [3]); the GLAV mapping states the existence of such a department in the global schema, even if its identifier is unknown (because it is not provided by \( V_1 \)). Therefore, John Doe is a certain answer to a query asking for all employees in IBM departments, based on the above GLAV mapping. This answer cannot be found using GAV mappings.

### 3 PROBLEM STATEMENT

In this section, we first formalize the notion of RDF integration system (Section 3.1). Then, we state the associated query answering problem (Section 3.2), for which Section 4 provides solutions.

#### 3.1 RDF integration system (RIS)

In an RDF integration system (RIS in short), data from heterogeneous sources, each of which may have its own data model and query language, is integrated into an RDF graph, consisting of an (RDFS) ontology and of data triples derived from the sources by means of GLAV-style mappings. Mappings allow (i) specifying the data made available from the sources, and (ii) organizing it according to the RIS ontology.

**Definition 3.1 (RIS mappings and extensions).** A RIS mapping \( m \) is of the form \( m = q_1(x) \leadsto q_2(x) \) where \( q_1 \) and \( q_2 \) are two queries with the same answer variables, and \( q_2 \) is a BGPQ whose body contains only triples of the forms:

- \((s, p, o)\) where \( p \in F_{\text{user}}\)
- \((s, r, c)\) where \( c \in F_{\text{user}}\)

The body of \( m \) is \( q_1 \) and its head is \( q_2 \). The extension of \( m \) is the set of tuples \( \text{ext}(m) = \{v_{m}(\delta(v_1), \ldots, \delta(v_n)) \mid \langle v_1, \ldots, v_n \rangle \in q_1(D)\} \), where \( q_1(D) \) is the answer set of \( q_1 \) on the data source \( D \) that \( m \) integrates and \( \delta \) is a function that maps source values to RDF values, i.e., IRIs, blank nodes and literals.

Intuitively, \( m \) specifies that the result of query \( q_1 \) on \( D \) transformed in RDF, i.e., the extension of \( m \), is exposed to the RIS as the result of the (BGP) query \( q_2 \).

**Example 3.2 (Mappings).** Consider the two mappings:

- \( m_1 \) with head \( q_1(x) \rightarrow (x, \text{ceoOf}, y), (y, r, \text{NatComp}) \) and
- \( m_2 \) with head \( q_2(x, y) \rightarrow (x, \text{hiredBy}, y), (y, r, \text{PubAdmin}) \)

Suppose that the body of \( m_1 \) returns \( (p_1^0) \) as its results, and that \( \delta \) function maps the value \( p_1^0 \) from the data source \( D_1 \) to the IRI \( p_1 \). Then, the extension of \( m_1 \) is: \( \text{ext}(m_1) = \{v_{m_1}(p_1)\} \). Furthermore, suppose that the body of \( m_2 \) returns \( (p_2^0, a^2_1) \), and that \( \delta \) maps the values \( p_2^0 \) and \( a^2_1 \) from the data source \( D_2 \) to the IRIs \( p_2 \) \( a_1 \). Then, the extension of \( m_2 \) is: \( \text{ext}(m_2) = \{v_{m_2}(p_2, a_1)\} \).

Given a set of RIS mappings \( M \), the extent of \( M \) is the union of the mappings’ extensions, i.e., \( E = \bigcup_{m \in M} \text{ext}(m) \), and we denote by \( \text{Val}(E) \) the set of values occurring in \( E \). We can now define the RIS data triples induced by some mappings and an extent thereof. These are all the data which is exposed (can be queried) through a RIS.

**Definition 3.3 (RIS data triples).** Given a set \( M \) of RIS mappings and an extent \( E \) of \( M \), the RIS data triples induced by \( M \) and \( E \) form an RDF graph defined as follows:

\[
G^M_E = \bigcup_{m=q_2 \Delta q_1 \in M} \{(\text{bgp2rdf}(\text{body}(q_2(x)), v_m(f)) \mid v_m(f) \in E)\}
\]

where

- \( \text{body}(q_2(x)) \) is the BGP body \( q_2 \) in which the answer variables \( x \) are bound to the values in the tuple \( v_m(f) \), part of \( E \);
- \( \text{bgp2rdf}(\cdot) \) is a function that transforms a BGP into an RDF graph, by replacing each variable with a fresh blank node.

Observe that, because we use GLAV mappings, RIS data triples may include fresh blank nodes, as exemplified below; these correspond to the existential variables allowed in GLAV mappings, as discussed at the end of Section 2.5.2.

**Example 3.4.** Reusing the mappings from Example 3.2, let \( M = \{m_1, m_2\} \) and its extent \( E = \{v_{m_1}(p_1), v_{m_2}(p_2, a)\} \). The RIS data triples they lead to are:

\[
G^M_E = \{(p_1, \text{ceoOf}, _b_1), (_b_1, r, \text{NatComp}), (p_2, \text{hiredBy}, _a_1), (a_1, r, \text{PubAdmin})\}
\]

In particular, the first and second triples contain the blank node \(_b_1\), introduced by \( \text{bgp2rdf} \) instead of the variable \( y \) in the head (query \( q_2 \) of \( m_1 \)). Importantly, only non-answer variables in a mapping head lead to blank nodes introduced this way: by Def. 3.3, answer variables (here \( x \) for \( m_1 \) and \( y \) for \( m_2 \)) are replaced with values from \( V_m(f) \), thus from \( \text{Val}(E) \).

Finally, we define a RIS as a tuple \( S = (O, R, M, E) \) stating that \( S \) allows to access (query), with the reasoning power given by the set \( R \) of RDFS entailment rules, the RDF graph comprising the ontology \( O \) and the data triples induced by the set of mappings \( M \) and their extent \( E \).

### 3.2 Query answering problem

The problem we consider is answering BGPQs in a RIS. We define certain answers in a RIS setting as follows:

**Definition 3.5 (Certain answer set).** The certain answer set of a BGPQ \( q \) on a RIS \( S = (O, R, M, E) \) is:

\[
\text{cert}(q, S) = \{\phi(x) \mid \phi \text{ homomorphism from body}(q) \to (O \cup G^M_E^R)\}
\]

where \( \phi(x) \) comprises only values from \( \text{Val}(E) \).

The certain answer set \( \text{cert}(q, S) \) is thus the subset of \( \{q \} \cup G^M_E^R \) restricted to tuples fully built from source values, i.e., we exclude tuples with blank nodes introduced by the mappings (see Def. 3.3). Note, however, that blank nodes can be exploited to answer queries, as shown below.

**Example 3.6 (Certain answers).** Consider the RIS \( S \) made of the ontology of GreenEdge in Example 2.2, the set \( R \) of entailment rules shown in Table 3, and the set of mappings \( M \) together with the extent \( E \) from Example 3.4.

Let \( q(x, y) \rightarrow (x, \text{worksFor}, y), (y, r, \text{Comp}) \) be the query asking "who works for which company", while the query \( q'(x, y) \rightarrow (x, \text{worksFor}, y), (y, r, \text{Comp}) \) asks "who works for some company". The only difference between them is \( y \) is an answer variable in \( q \) and not in \( q' \). The certain answer set of \( q \) is \( 0, \)
while the certain answer set of \( q \)’ is \( \{ (p_1) \} \). This answer results from the RIS data triples \( (p_1, \text{worksFor}, \_b_2), (\_b_2, \tau, \text{NatComp}) \), which are entailed from:

- the \( G^M \) triples \( (p_2, \text{ceoOf}, \_b_4), (\_b_4, \tau, \text{NatComp}) \), with the blank node \( \_b_4 \) discussed in Example 3.4, and:
- either the \( O \) triples \((\text{ceoOf}, \_sp, \text{worksFor}), (\_sp, \text{NatComp})\) together with the \( \mathcal{R} \) rules \( \text{rdfs3}, \text{rdfs5} \), or the \( O \) triples \((\text{ceoOf}, \_sp, \text{worksFor}), (\text{NatComp}, \_sc, \text{Comp})\) together with the \( \mathcal{R} \) rules \( \text{rdfs3}, \text{rdfs5} \).

The query \( q \) has no answer because it requires a value not available from the source: the company for which \( p_1 \) works; the RIS only knows the existence of such value through the blank node \( \_b_2 \) gotten by \( \text{bgp2rdf} \) in its data triples. In contrast, \( q \) allows finding out that \( p_1 \) works for (as CEO of) some (national) company, even though the mapping \( m_1 \) (the only one involving companies) does not expose the company IRI through the RIS.

The problem we study in the next section is:

**Problem 1.** Given a RIS \( S \), compute the certain answer set of a BGPQ \( q \) on \( S \), i.e., \( \text{cert}(q, S) \).

### 4 QUERY ANSWERING IN A RIS

Since we adopt a mediator-style approach, the RIS data triples \( G^M \) are not materialised, hence the satisfaction of \( O \cup G^M \) cannot be computed to answer queries as defined above. Instead, queries are rewritten in terms of the remote heterogeneous sources, based on the RIS ontology \( O \), reasoning power \( \mathcal{R} \) and mappings \( M \). We present three query answering strategies, which differ in how reasoning performed at query time, as outlined in Figure 2.

#### All reasoning at query time

The first strategy will be detailed in Section 4.1. First, it reduces the RIS query answering problem to standard query evaluation in an RDF data management system, by reformulating (step (1) in Figure 2) the query \( q \) on the RIS ontology \( O \) and entailment rules \( \mathcal{R} = \mathcal{R}_e \cup \mathcal{R}_a \).

The obtained reformulated query \( Q_{\mathcal{E},a} \) thus yields the expected certain answers when evaluated on the RIS data triples (recall Section 2.4), provided that answers with blank nodes introduced by the \( \text{bgp2rdf} \) function are discarded (recall Section 3.2). Since these data triples are not materialized, the RDF query evaluation problem is then reduced to relational \textit{view-based} query answering, by rewriting \( Q_{\mathcal{E},a} \) using the RIS GLAV mappings \( M \) seen as LAV views (step (2)). This produces a relational rewriting \( q_r \) over the mappings extension (step (3)), whose evaluation with a mediator query engine provides the desired certain answers (steps (4) and (5)).

#### Some reasoning at query time

The second strategy (detailed in Section 4.2) is a main contribution of this paper. First, it reduces the RIS query answering problem to \textit{saturation-based} query answering by reformulating (step (1”)) the query \( q \) based on \( O \) and \( \mathcal{R}_e \) only (not \( \mathcal{R} = \mathcal{R}_e \cup \mathcal{R}_a \) as above). The obtained reformulation \( Q_{\mathcal{E}} \) thus yields the expected certain answer set when evaluated on the RIS data triples \textit{saturated} with \( \mathcal{R}_a \) (recall Section 2.4), again provided that the answers with blank nodes introduced by the \( \text{bgp2rdf} \) function are discarded (as above). Since these triples are not materialized in a RIS, hence cannot be saturated with \( \mathcal{R}_a \), the saturation-based query answering problem is in turn reduced to relational \textit{view-based} query answering, by rewriting \( Q_{\mathcal{E}} \) using the RIS GLAV mappings \textit{saturated} \( O \) and \( \mathcal{R}_a \), seen as LAV views. These saturated mappings, denoted \( M^{s,O} \), are obtained (step (A)) from the original ones by adding to their head queries \( q_s \) all the implicit data triples they model w.r.t. \( O \) and \( \mathcal{R}_a \). Then, the partially reformulated query \( Q_{\mathcal{E}} \) is rewritten using \( M^{s,O} \) (step (2’)) and the resulting query (step (3)) is evaluated as in the first strategy (steps (4) and (5)). Importantly, mappings are saturated offline, and need to be updated only when some mapping changes. This limits both the reasoning effort at query time and the complexity of the reformulated query to rewrite, hence the rewriting time needed to obtain a rewriting \( q_{\text{rew}} \) over the data sources, as our experiments show (Section 5).

#### No reasoning at query time

Finally, the third strategy (detailed in Section 4.3) reduces the RIS query answering problem directly to view-based query answering. Here, the mappings are saturated offline as above (step (A)), in order to model all explicit and implicit RIS data triples. Also, these mappings are complemented offline as above (step (A)), in order to model all explicit and implicit RIS schema triples w.r.t. \( O \) and \( \mathcal{R}_a \); since only \( \mathcal{R}_a \) rules entail new schema triples (Table 3), \( O^R \) is actually equal to \( O^\mathcal{R}_a \). This second set of mappings is also computed offline, and only needs to be updated when the ontology changes. A query \( q \) just needs to be rewritten based on the above mappings \( M^{s,O} \cup M^{O\mathcal{R}_a} \), seen as LAV views (step (2”)), in order to obtain, as above, a rewriting \( q_{\text{rew}} \) over the data sources (step(3’), followed by the evaluation steps (4) and (5)).

Before going into the technical details of the above strategies, we introduce a set of simple functions. The \( \text{bgp2cu} \) function transforms a BGP into a conjunction of atoms with ternary predicates \( T \) (standing for "triple") as follows: \( \text{bgp2cu}(\{ (s_1, p_1, o_1), \ldots, (s_n, p_n, o_n) \}) = T(s_1, p_1, o_1) \wedge \cdots \wedge T(s_n, p_n, o_n) \). The \( \text{bgpq2cu} \) function transforms a BGPQ \( q(x) \leftarrow \text{body}(q) \) into a CQ \( q(x) \leftarrow \text{bgpq2cu}(\text{body}(q)) \). Finally, the function \( \text{ubgpq2ucu} \) function transforms a \( \text{UBGPQ} \bigcup_{i=1}^n q_i(x_i) \) into a UCQ by applying the above \( \text{bgpq2cu} \) function to each of its \( q_i \).

#### 4.1 Rewriting Fully-Reformulated Queries using Mappings as Views: \( \text{REW-CA} \)

Based on [12], the first step of this strategy, (1) in Figure 2, reformulates a query \( q \) w.r.t. \( O \) and \( \mathcal{R} = \mathcal{R}_e \cup \mathcal{R}_a \) into a query \( Q_{\mathcal{E},a} \). This allows obtaining the certain answers directly from the RIS data triples, and not from their saturation after they have been
augmented with $O$ (recall Definition 3.5). Indeed, the correctness of the reformulation ensures that the certain answers of $q$ on the RIS $S$ correspond precisely to those of $Q_{c,a}$ asked on $S$ when disregarding $O$ and $R$, as formally expressed in the next lemma. Of course, this still does not provide a concrete solution to obtain the desired certain answers using standard query evaluation, since the RIS data triples $G^M$ are not materialized.

**Lemma 4.1.** Let $S = (O, R, M, E)$ be a RIS, $q$ be a BGPQ and $Q_{c,a}$ its UBGPQ reformulation w.r.t. $O, R = R_e \cup R_a$ using [12]. Then:

\[
\text{cert}(q, S) = \text{cert}(Q_{c,a}, (\emptyset, \emptyset, M, E))
\]

The proof of this and our following claims can be found in [13]. Recall that the RIS data triples are defined from the mappings $M$ by, for every mapping $m = q_1(x) \rightarrow q_2(x) \in M$, (i) evaluating the mapping body $q_1(x)$ on the data source to produce its extension $\text{ext}(m) \in E$, and then (ii) instantiating the mapping head $q_2(x)$ with its extension. At the same time, this is also how the instance of a data integration system based on LAV views and their extensions is defined in a relational setting (Section 2.5.1). Based on this analogy, we recast the RIS query answering problem of the above Lemma, into a relational view-based query answering one. To this aim, we treat our mappings as LAV views:

**Definition 4.2 (Mappings as relational LAV views).** Let $m = q_1(x) \rightarrow q_2(x)$ be a mapping. Its corresponding relational LAV view is:

\[
V_m(x) \leftarrow \text{body}(q_2(x)).
\]

**Example 4.3.** The relational LAV views corresponding to the mappings $m_1, m_2$ from Example 3.2 are:

- $V_{m_1}(x) \leftarrow T(x, \text{CEOOf}, y), T(y, r, \text{NatComp})$
- $V_{m_2}(x, y) \leftarrow T(x, \text{hiredBy}, y), T(y, r, \text{PubAdmin})$

We denote the set of views derived from all the mappings $M$ by Views$(M)$. Crucially, the extent $E$ of the mapping set $M$ is also an extent for the corresponding set of views Views$(M)$. Based on the above Lemma 4.1, treating mappings and their extent as relational LAV views and their extent, and seeing (UB)GPQs as (U)CQs with the help of the functions introduced in the beginning of Section 4, we reduce the RIS query answering problem to view-based query answering:

**Theorem 4.4 (REW-CA correctness).** Let $S = (O, R, M, E)$ be a RIS and $q$ be a BGPQ. Let $Q_{c,a}$ be the reformulation of $q$ w.r.t. $O$ and $R$ using [12]. Then:

\[
\text{cert}(q, S) = \text{cert}(Q_{c,a}, \text{Views}(M), E)
\]

where $\text{cert}(\text{ubgpq2ucq}(Q_{c,a}), \text{Views}(M), E)$ denotes the certain answer set of $\text{ubgpq2ucq}(Q_{c,a})$ over Views$(M)$ and $E$.

Importantly, this provides an effective solution to RIS query answering problem by using state-of-the-art view-based query rewriting techniques [31], in particular for step (2) in Figure 2.

**Example 4.5 (REW-CA query answering).** Consider again the RIS in Example 3.6 and the query $q(x, y) \equiv (x, y, z), (z, r, t), (y, \text{CEO}, \text{Corp}), (x, \text{worksFor}, a), (a, r, \text{PubAdmin})$ asking “who works for some public administration, and what working relationship he/she has with some company". Its UBGPQ reformulation, seen as a UCQ, is shown in Figure 3. Its maximally-contained rewriting based on the views obtained from the RIS mappings is: $q_1(x, \text{CEOOf}) \leftarrow \text{body}(q_2(x)), q_1(x, y, z), y \text{ obtained from the second CQ in the above union. This becomes clear when the views are replaced by their bodies: } q_1(x, \text{CEOOf}) \leftarrow T(x, \text{CEOOf}, y), T(y, r, \text{NatComp}), T(x, \text{hiredBy}, y), T(y, r, \text{PubAdmin})$. Note $Q_{c,a} = q_1(x, \text{CEOOf}) \leftarrow T(x, \text{CEOOf}, z), T(x, t, \text{NatComp}), T(x, \text{WorksFor}, a), T(a, r, \text{PubAdmin})$ union $q_1(x, \text{CEOOf}) \leftarrow T(x, \text{CEOOf}, z), T(z, r, \text{NatComp}), T(x, \text{hiredBy}, a), T(a, r, \text{PubAdmin})$ union $q_1(x, \text{CEOOf}) \leftarrow T(x, \text{hiredBy}, z), T(z, r, \text{NatComp}), T(x, \text{CEOOf}, a), T(a, r, \text{PubAdmin})$ union $q_1(x, \text{hiredBy}) \leftarrow T(x, \text{hiredBy}, z), T(z, r, \text{NatComp}), T(x, \text{CEOOf}, a), T(a, r, \text{PubAdmin})$ union $q_1(x, \text{hiredBy}) \leftarrow T(x, \text{hiredBy}, z), T(z, r, \text{NatComp}), T(x, \text{CEOOf}, a), T(a, r, \text{PubAdmin})$.

**Figure 3:** Sample reformulation for Example 4.5.

that the other CQs cannot be rewritten given the available views. With the current RIS, this rewriting yields an empty certain answer set to $q$, i.e., $\text{cert}(q, S) = \emptyset$, because the extent of the mappings, hence of the views, is: $E = (V_{m_1}(q_1), V_{m_2}(q_2), \ldots)$. However, if we add $V_{m_1}(q_1, q_2, \ldots)$ to $E$, then $\text{cert}(q, S) = \{(q_1, \text{CEOOf})\}$.

**4.2 Rewriting Partially-Reformulated Queries using Saturated Mappings as Views: Rew-C.**

In contrast with the REW-CA strategy that performs all the reasoning w.r.t. $O$ and $R = R_e \cup R_a$ at query time, our second strategy called Rew-C splits the reasoning work between offline preprocessing and query time. The first step of this strategy, labeled (1') in Figure 2, reformulates a query $q$ using [12], but solely w.r.t. $O$. $Q_{c,a}$, producing a UBGPQ denoted $Q_c$. From the correctness of this reformulation step, and the fact that only $R_a$ needs to be considered to answer $Q_c$, with respect to the entire set of rules $R$ (recall Section 2.4), the certain answer set of $q$ asked on the RIS $S$ is exactly the certain answer set of $Q_c$ asked on $S$ when disregarding $R_e$. Formally:

**Lemma 4.6.** Let $S = (O, R, M, E)$ be a RIS, $q$ be a BGPQ and $Q_c$ its reformulation w.r.t. $O, R_e \cup R_a$ [12]. Then:

\[
\text{cert}(q, S) = \text{cert}(Q_c, (O, R_e, M, E))
\]

In other words, the desired answer set could be obtained by evaluating $Q_c$ on the RIS data triples $G^M$ saturated by $R_e$. Again, since the RIS data triples are not materialized, this does not provide a concrete solution. To account for the impact of the ontology $O$ and the entailment rules $R$ on these "virtual" data triples, we rely on BGPQ saturation [25]: given a BGPQ $q, O$ and $R$, the saturation $q^{R,O}$ is $q$ augmented with all the triples $q$ implicitly asks for given the ontology $O$ and the rules $R$. BGPQ saturation is exemplified below:

**Example 4.7 (BGPQ saturation).** Consider the ontology $O$ of $G_a$ and the query $q(x) \equiv (x, \text{hiredBy}, y), (y, r, \text{NatComp})$ asking who has been hired by a national company. Its saturation w.r.t. $R_a$, $O$ is: $q^{R_a,O}(x) \equiv \text{body}(q(x)), (x, \text{worksFor}, y), (x, r, \text{Person}), (y, r, \text{Corp}), (y, r, \text{Org})$.

We use BGPQ saturation to saturate the RIS mapping heads w.r.t. $R_a$, $O$, so that the saturated mappings together with $E$ model the saturated RIS data triples w.r.t. $R_a, O$. To compute $q^{R_a,O}$ we (1) saturate $q(O) \cup O$ using $R_a$, then (2) add to the body of $q$ all triples thus inferred.

**Definition 4.8 (Mappings saturation).** The saturation of a set $M$ of RIS mappings w.r.t. entailment rules $R_a$ and ontology $O$ is:

\[
M^{a,O} = \bigcup_{m \in M} q_1(x) \rightarrow q_2^{R_a,O}(x) \mid m \equiv q_1(x) \rightarrow q_2(x)
\]

305
We saturate mappings offline, and just need to update them when \( O \) or the mapping heads change.

**Example 4.9 (Saturated mappings).** Consider the RIS of Example 3.6, the mapping heads in \( M^{a,O} \) are (added implicit triples are in blue):

\[
m_1 : \quad q_1^{a,O}(x) \leftarrow (x, \text{ceeOf}, y), (y, r, \text{NatComp})
\]

\[
m_2 : \quad q_2^{a,O}(x, y) \leftarrow (x, \text{hiredBy}, y), (y, r, \text{PubAdmin})
\]

From the above Lemma and the use of saturated RIS mappings instead of the original ones, we show:

**Lemma 4.10.** Let \( S = \langle O, R, M, E \rangle \) be a RIS, \( q \) be a BG PQ and \( Q_c \) its reformulation w.r.t. \( O, R_c \) [12]. Then:

\[
cert(q, S) = cert(Q_c, \langle O, \emptyset, M^{a,O}, E \rangle)
\]

This result allows solving the RIS query answering problem by relational view-based query rewriting (step (2') in Figure 2):

**Theorem 4.11 (REW-C correctness).** Let \( S = \langle O, R, M, E \rangle \) be a RIS, \( q \) be a BG PQ and \( Q_c \) its reformulation w.r.t. \( O, R_c \). Then:

\[
cert(q, S) = cert(bgp2ucq(Q_c), Views(M^{a,O}, E))
\]

**Example 4.12 (REW-CA).** Consider again the RIS in Example 3.6 and the query \( q \) of Example 4.5. Its reformulation \( Q_c \) w.r.t. \( O, R_c \), seen as a UCQ, is:

\[
q(x, \text{ceeOf}) \leftarrow T(x, \text{ceeOf}, z), T(z, r, \text{NatComp})
\]

\[\cup q(x, \text{hiredBy}) \leftarrow T(x, \text{hiredBy}, z), T(z, r, \text{PubAdmin})
\]

This reformulation is therefore rewritten using the RIS views as:

\[
q_1(x, \text{ceeOf}) \leftarrow V_m(x, y)
\]

\[
q_2(x, \text{hiredBy}) \leftarrow V_m(x, y)
\]

This strategy does not reason at query time at all. Instead, it rewrites a query \( q \) based on the saturated RIS mappings \( M^{a,O} \) as above, and on a specific set of ontology mappings we build to model the saturated RIS ontology as a data source:

**Definition 4.13 (Ontology mappings).** The set of ontology mappings for a RIS ontology \( O \) is:

\[
M^{O} = \bigcup_{s \in \{\text{a}, \text{b}, \text{c}, \text{d}, \text{e}, \text{f}, \text{g}, \text{h}, \text{i}, \text{j}, \text{k}, \text{l}, \text{m}, \text{n}, \text{o}, \text{p}, \text{q}, \text{r}, \text{s}, \text{t}, \text{u}, \text{v}, \text{w}, \text{x}, \text{y}, \text{z} \}} \{m_x | m_x = q_1(s, o) \rightarrow q_2(s, o)\}
\]

The extension of an ontology mapping \( m_x \) is \( ext(m_x) = \{V_m(s, o) | (s, x, o) \in O^{R_c}\} \). The extent of \( M^{O} \) is denoted \( E^{O}_C \).

We compute ontology mappings offline, and only need to update them when the ontology changes. The ontology mapping extensions \( E^{O}_C \) store all the explicit and implicit RIS ontology triples (recall from Section 2.2 that only \( R_c \) lead to such triples). Importantly, this leads to the observation that a query triple that refers to the ontology (schema) can be evaluated on the ontology mapping extensions alone. Formally:

\[
q(x, \text{ceeOf}) \leftarrow V_m(x, y)
\]

\[
\cup q(x, \text{hiredBy}) \leftarrow V_m(x, y)
\]

\[
\cup q(x, \text{hiredBy}) \leftarrow V_m(x, y)
\]

\[
\cup q(x, \text{hiredBy}) \leftarrow V_m(x, y)
\]

**Figure 4: Sample rewriting for Example 4.17.**

**Lemma 4.14.** Let \( S = \langle O, R, M, E \rangle \) be a RIS and \( q \) be a BG PQ. Then:

\[
cert(q, S) = cert(q, \langle O, R, M^{O}, E^{O}_C, E_C \rangle)
\]

This lemma effectively "pushes" \( R_c \) reasoning in the set of mappings (to which we add \( M^{O} \)) and the extent (to which we add \( E^{O}_C \)). Next, we rely (as we did for REW-CA) on mappings saturation with \( O, R_a \) to also push \( R_a \) reasoning in the mappings, leading to:

**Lemma 4.15.** Let \( S = \langle O, R, M, E \rangle \) be a RIS and \( q \) be a BG PQ. Then:

\[
cert(q, S) = cert(q, \langle O, R, M^{O}, E^{O}_C, E_C \rangle)
\]

This allows to reduce RIS query answering to relational view-based query rewriting (step (2') in Figure 2):

**Theorem 4.16 (REW correctness).** Let \( S = \langle O, R, M, E \rangle \) be a RIS and \( q \) be a BG PQ. Then:

\[
cert(q, S) = cert(bgp2ucq(q), Views(M^{O}, E^{O}_C, E_C))
\]

**Example 4.17 (REW).** Consider again the RIS in Example 3.6 and the query \( q \) of Example 4.5 seen as a CQ:

\[
q(x, y) \leftarrow T(x, y, z), T(z, r, t), T(y, \text{sp}, \text{worksFor}),
\]

\[
T(z, \text{sp}, \text{Comp}), T(x, \text{worksFor}, a),
\]

\[
T(a, r, \text{PubAdmin})
\]

Its maximally-contained rewriting \( q_{rew} \) based on the views obtained from the RIS saturated mappings and ontology mappings appears in Figure 4. This rewriting is much larger than the ones of the two preceding techniques: this is due to the ontology mappings. If we assume that \( E \) also contains \( V_m(p, a) \), as we did in
Example 4.5, we obtain again \( \text{cert}(q, S) = \{ \langle p_1, \text{ceoOf} \rangle \} \), which results from the evaluation of the first CQ in the UCQ rewriting; the other CQs yield empty results because some required \( <_{sc} \) or \( <_{sp} \) constraints are not found in the views built from the RIS ontology mappings.

**How do our strategies compare?** Since they are all correct, they lead to the same RIS certain answer set, however they do not necessarily compute the same view-based rewritings. Indeed, \( \text{REW} \) considers the additional set \( M_{\text{O}} \) of ontology mappings. Hence, for queries over the ontology, i.e., featuring in a property position \( <_{sc}, <_{sp}, \langle \cdot, \cdot \rangle, \rightarrow, \text{or} \), a variable, \( \text{a} \text{REW} \) rewriting is larger than a \( \text{REW-CA} \) or \( \text{REW-C} \) rewriting and, to be answered, requires the additional ontology source. In contrast, \( \text{REW-CA} \) and \( \text{REW-C} \) yield logically equivalent rewritings; we minimize them both to avoid possible redundancies, thus they become identical (up to variable renaming). Hence, \( \text{REW-CA} \) and \( \text{REW-C} \) do not differ in how these rewritings are evaluated. Instead, they differ in how the rewritings are computed, or, equivalently, on the distribution of the reasoning effort on the data and mappings, across various query answering stages. As our experiments show, given the computational complexity of view-based query rewriting [42], this difference has a significant impact on their performance.

5 EXPERIMENTAL EVALUATION

We now describe our experiments with RIS query answering. In addition to our strategies based on query rewriting, we include in our comparison a simple alternative strategy, based on materialization and denoted \( \text{MAT} \). Offline (before answering queries), this strategy materializes the RIS data triples and saturates them with the rule set \( \mathcal{R} \). The materialization is stored and saturated in an RDF data management system (RDFDB, in short). Then, \( \text{MAT} \) query answering amounts to query evaluation on the saturated materialization. Therefore, \( \text{MAT} \) query answering can be seen as a lower bound for query answering through other strategies.

5.1 Experimental settings

**Software** Our platform is developed in Java 1.8, as follows. Our RDFDB is OntoSQL\(^2\), a Java platform providing efficient RDF storage, saturation, and query evaluation on top of an RDBMS [14, 29], relying on Postgres v9.6. To save space, OntoSQL encodes IRIs and literals into integers, and a dictionary table which allows going from one to the other. It stores all resources of a certain type in a one-attribute table, and all (subject, object) pairs for each property (including RDFS schema properties) in a table; the tables are indexed. OntoSQL is used in the \( \text{MAT} \) strategy, and it also provides the RDF query reformulation algorithm [12].

We rely on the Graal engine [9] for view-based query rewriting. Graal is a Java toolkit dedicated to query answering algorithms in knowledge bases with existential rules (a.k.a. tuple-generating dependencies). Since the relational view \( \text{vm}(x) \leftarrow b p z c a t(b o d y(q_2)) \) corresponding to a GLAV mapping \( m \) (recall Def. 4.2) can be seen as a specific existential rule of the form \( \text{vm}(x) \leftarrow b p z c a t(b o d y(q_2)) \), the query reformulation algorithm of Graal can be used to rewrite the UCQ translation of a BGPQ with respect to a set of RIS mappings. To execute queries against heterogeneous data sources, we use Tatoonie [4, 10], a Java-based mediator (or polystore) system, capable both of pushing queries in underlying data sources and (unlike other polystores, e.g., [24]) of evaluating joins within the mediator engine. Query rewritings produced by Graal are unfolded into queries on the

---

\(^2\)https://ontosql.inria.fr

---

\(^3\)https://downloads.sourceforge.net/project/bsbmtools/bsbmtools/bsbmtools-0.2

5.2 Experimental scenarios

**RDF Integration Systems (RIS) used** Our first interest was to study scalability of RIS query answering, in particular in the relational setting studied in many prior works. To achieve this, we used the BSBM benchmark relational data generator\(^4\) to build databases consisting of 10 relations named producer, product, offer, review etc. Using two different benchmark scale factors, we obtained a data source DS\(_1\) of 154,054 tuples across the relations, respectively, DS\(_2\) of 7,843,660 tuples; both are stored in Postgres. We used two RDFs ontologies \( O_1 \) respectively \( O_2 \), containing, first, subclass hierarchies of 151 (resp. 201) product types, which come with DS\(_1\), respectively; DS\(_2\). To \( O_1 \) and \( O_2 \), we add a natural RDFS ontology for BSBM composed of 26 classes and 36 properties, used in 40 subclass, 32 subproperty, 42 domain and 16 range statements.

**Relational-sources RIS** We devised two sets \( M_1, M_2 \) of 307, respectively. 3863 mappings, which expose the relational data from DS\(_1\), respectively, DS\(_2\) as RDF graphs. The relatively high number of mappings is because: (i) each product type (of which there are many, and their number scales up with the BSBM data size) appears in the head of a mapping, enabling fine-grained and high-coverage exposure of the data in the integration graph; (ii) we also generated more complex GLAV mappings, partially exposing the results of join queries over the BSBM data; interestingly, these mappings expose incomplete knowledge, in the style of Example 3.4.

The mapping sets lead to the RIS graphs of \( 2 \cdot 10^8 \), respectively. 108 \( \cdot 10^6 \) triples. Their saturated versions comprise respectively \( 3.4 \cdot 10^9 \) and \( 185 \cdot 10^8 \) triples. Our first two RIS are thus: \( S_1 = \langle O_1, R, M_1, E_1 \rangle \) and \( S_2 = \langle O_2, R, M_2, E_2 \rangle \), where \( E_i \) for \( i \) in \( \{1,2\} \) are the extents resulting from DS\(_1\) and \( M_i \).

**Heterogeneous-sources RIS** Second, going beyond relational-sources OBDA [16, 17, 44], our architecture extends to heterogeneous data sources. For that, we converted a third (33\%) of DS\(_1\), DS\(_2\) into JSON documents, and stored them into MongoDB, leading to the JSON data sources denoted DS\(_{1j}\), DS\(_{2j}\); the relational sources DS\(_{1r}\), DS\(_{2r}\) store the remaining (relational) data. Conceptually, for \( i \) in \( \{1,2\} \), the extension based on DS\(_{ir}\) and extension based on DS\(_{jr}\) form a partition of \( E_i \). We devise a set of JSON-to-RDF mappings to expose DS\(_{1j}\) and DS\(_{2j}\) into RDF, and denote \( M_j \) the set of mappings exposing DS\(_{1j}\), DS\(_{2j}\), together, as an RDF graph; similarly, the mappings \( M_k \) expose DS\(_{1r}\) and DS\(_{2r}\) as RDF. Our last two RIS are thus: \( S_1 = \langle O_1, R, M_1, E_1 \rangle \) and \( S_2 = \langle O_2, R, M_2, E_2 \rangle \), where \( E_3 \) is the extent of \( M_3 \) based on DS\(_{r1}\) and DS\(_{r1}\), while \( E_4 \) is the extent of \( M_4 \) based on DS\(_{r2}\) and DS\(_{r2}\). The RIS data and ontology triples of \( S_1 \) and \( S_3 \) are identical; thus, the difference between these two RIS is only due to the heterogeneity of their underlying data sources. The same holds for \( S_2 \) and \( S_4 \).

**Queries** We devised a set of 28 BGP queries having from 1 to 11 triple patterns (5.5 on average), of varied selectivity (they return between 2 and 330 \( \cdot 10^4 \) results in \( S_1 \) and \( S_3 \) between 2 and 4.4 \( \cdot 10^8 \) results in \( S_2 \) and \( S_4 \)); 6 among them query the data and the ontology (recall Example 2.6), a capability which most

---

\(^4\)https://downloads.sourceforge.net/project/bsbmtools/bsbmtools/bsbmtools-0.2

---

307
Table 4: Characteristics of the queries used in our experiments.

<table>
<thead>
<tr>
<th>RIS</th>
<th>Q01</th>
<th>Q01a</th>
<th>Q01b</th>
<th>Q02</th>
<th>Q02a</th>
<th>Q02b</th>
<th>Q02c</th>
<th>Q03</th>
<th>Q04</th>
<th>Q07</th>
<th>Q07a</th>
<th>Q09</th>
<th>Q10</th>
<th>Q13</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Q01</td>
<td>7</td>
<td>21</td>
<td>175</td>
<td>21</td>
<td>49</td>
<td>147</td>
<td>1225</td>
<td>525</td>
<td>1</td>
<td>5</td>
<td>19</td>
<td>2</td>
<td>3</td>
<td>5617</td>
</tr>
<tr>
<td>Q01a</td>
<td>1272</td>
<td>4376</td>
<td>22738</td>
<td>16</td>
<td>56</td>
<td>174</td>
<td>1342</td>
<td>19</td>
<td>91</td>
<td>2</td>
<td>3</td>
<td>5617</td>
<td>9</td>
<td>13190</td>
</tr>
<tr>
<td>Q01b</td>
<td>21</td>
<td>175</td>
<td>1407</td>
<td>63</td>
<td>147</td>
<td>525</td>
<td>1225</td>
<td>4375</td>
<td>1</td>
<td>5</td>
<td>19</td>
<td>7</td>
<td>9550</td>
<td>84</td>
</tr>
<tr>
<td>Q02</td>
<td>5514</td>
<td>11793</td>
<td>863729</td>
<td>124</td>
<td>598</td>
<td>1058</td>
<td>1570</td>
<td>5</td>
<td>4487</td>
<td>2</td>
<td>3</td>
<td>299902</td>
<td>10</td>
<td>16776</td>
</tr>
<tr>
<td>Q02a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q02b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q02c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q03</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q07</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q07a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q09</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5: Query answering times on the smaller RIS $S_1$ (top, relational sources) and $S_3$ (bottom, heterogeneous sources).

Figure 6: Query answering times on the larger RIS $S_2$ (top, relational sources) and $S_4$ (bottom, heterogeneous sources).

Section 4.3, we noted that the size of the rewriting produced by REW is larger (by a multiplicative factor of 29 to 74 in $S_1$ and $S_3$, and of 33 to 969 in $S_2$ and $S_4$) than the rewritings of the two other strategies, which led to an explosion of the time spent minimizing the rewriting, and made REW overall unfeasible; the details of these tests can be found online\(^4\). On queries that do not carry over the ontology, REW produces the same rewritings as the other methods. Thus, we do not report further REW performance below.

**Query answering time comparison** Figure 5 depicts the query answering times, on the smaller RIS, of REW-CA, REW-C and MAT. The size of (number of BGPQs in) the reformulation of each query w.r.t. $\mathcal{R}$, $|Q_{ca}|$ appears in parentheses after the query name, in the labels along the $x$ axis. Given that $S_1$, $S_3$ have the same RIS data triples, the MAT strategy coincides among these two RIS. Figure 6 shows the corresponding times for the largest RIS $S_2$ and $S_4$, the same observations apply. Note the logarithmic time axes.

A first observation is that our query set is quite diverse; their answering times range from a few to more than 10\(^6\) ms.

As expected, query answering in MAT is the fastest in most cases, since it has no reasoning work to do at query answering time. However, it required, for $S_1$, $S_2$, 1.2 · 10\(^5\) ms to build the materialization and 1.49 · 10\(^5\) ms more to saturate it, whereas for $S_3$, $S_4$, these times are 14h46 (5.31 · 10\(^7\) ms), respectively, 1h28 (5.28 · 10\(^6\) ms).

---

\(^4\)Experiment web site: https://gitlab.inria.fr/mburon/org/blob/master/projects/het2onto-benchmark/bshm/
ms). Not only these are orders of magnitude more than all query answering times; recall also that materializing $G^M$ requires maintaining it when the underlying data changes, and its saturation $(G^M \cup O)^S$ needs a second level of maintenance. Thus, MAT is not practical when data sources change. We were surprised to see REW-C and REW-CA somehow faster than MAT for queries $Q_{20}$ and $Q_{24}$. Answering these queries through MAT within OntoSQL leads to producing many results that involve mapping-generated blank nodes, tuples which should not appear in our certain answers, as per Definition 3.5. We remove such tuples in post-processing mode, which leads to a performance overhead for MAT. REW-C and REW-CA, in contrast, are answered by evaluating rewritings, and do not have to apply such a result pruning. It remains to be seen if this pruning could be pushed in an RDFDB; note that not all answers including blank nodes should be pruned, only those whose blank nodes are due to mappings.

In each scenario, we observe that REW-C is faster or takes as long as REW-CA. Since the two approaches produce the same rewritings, the difference is due to steps before the step (3) in Figure 2. It turns out it is due to the rewriting time, which in turn strongly depends on the size of the reformulation it receives as input. In REW-C, the reformulations w.r.t. $\mathcal{R}_a$ are of size 1 (no union, just one BGP) for queries on data triples only, and never exceed 64 in $S_1$ and $S_3$ and 200 in $S_2$ and $S_4$. Whereas, in REW-CA the reformulation sizes are much larger. REW-C is most often faster than REW-CA, by up to two orders of magnitude e.g., for $Q_{20a}, Q_{19}$ and $Q_{20a}$ on $S_2$, the latter two on $S_4$ etc. One order of magnitude speed-up is noticeable even on the smaller RIS $S_1$ (Figure 5) for $Q_{20a}$. As a consequence, REW-C completes successfully in all scenarios we study, whereas REW-CA fails to complete for many queries with timeout set to 10 min (missing yellow bars in Figure 6), in close correlation with the increased number of reformulations.

**Scaling in the data size** As stated in Section 5.2, there is a scale factor of about 50 between $S_1$, $S_3$ on one hand, and $S_2$, $S_4$ on the other. Figures 5 and 6 show that the query answering times generally grow by less than 50, when moving from $S_1$ to $S_2$, and from $S_3$ to $S_4$. This is mostly due to the good scalability of PostgreSQL (in the all-relational RIS), TatooinE (itself building on PostgreSQL and MongoDB, in the heterogeneous RIS), and OntoSQL (for MAT). As discussed above, computation steps we implemented outside these systems are strongly impacted by the mappings, ontology and query; intelligently distributing the reasoning effort, as REW-C does, avoids the heavy performance penalties that from which REW-CA and REW sometimes suffer.

**Impact of heterogeneity** REW-CA and REW-C incur a (modest) overhead when combining data from PostgreSQL and MongoDB (heterogeneous RIS) w.r.t. the relational-sources RIS. Part of this is due to the cost of marshalling data across system boundaries; the rest is due to imperfect optimization within TatooinE. Overall, the comparison demonstrates that RIS query answering is feasible and quite efficient even on heterogeneous data sources.

### 5.4 Experiment conclusion

In a setting where the data, ontology and mappings do not change, MAT is an efficient and robust query answering technique, at a rather high cost to materialize and saturate the RIS instance. In contrast, in a dynamic setting, REW-C smartly combines partial reformulation and view-based query rewriting to efficiently compute query answers. The changes it requires when the ontology and mappings change (basically re-saturating mapping heads) are light and likely to be very fast. Thus, we conclude that REW-C is the best query answering strategy for dynamic RIS.

### 6 RELATED WORK AND CONCLUSION

Ontologies have been used to integrate relational or heterogeneous data sources in mediators [49] with LAV views based on description logics [1, 37] or their combination with Datalog [28, 30]. Semantics have been used at the integration level since e.g., [20] for SGML and soon after for RDF [6, 7]; data is considered represented and stored in a flexible object-oriented model, thus no mappings are used.

Our work follows the OBDA paradigm introduced in [41]. This paradigm was conceived to enhance access to relational data by mappings to an ontology expressed in a dialect of the DL-Lite description logic family (typically DL-Lite with underpinning the OWL 2 QL profile of the W3C ontological language OWL 2). Mature DL-based systems include Mastro$^2$ [17] and Ontop$^6$ [16, 43]. Another notable OBDA system, namely Ultrawrap$^6$OBDA [44], is based on an extension of RDFS to inverse and transitive properties. All these systems rely on GAV mappings.

Compared to these, our main novelty is to handle GLAV mappings and provide query answering algorithms for the resulting novel RIS setting. Note that formal OBDA frameworks with GLAV mappings have long been defined, e.g., in [18], but not put into practice. Regarding the other components of OBDA, we consider a simpler ontological language than existing OBDA systems, but support BGPQs on both data and ontological triples, a feature hardly found in these systems (an exception is [33]).

As explained in the introduction, GLAV mappings maximize the expressive power of the integration system. In particular, they allow to expose a form of incomplete information (recall Example 3.6). To some extent, GLAV mappings may be simulated by GAV mappings provided with so-called Skolem functions on answer variables, as suggested for instance in [21]. To illustrate, consider the GLAV mapping $m_1 = q_1(x) \leadsto q_2(x)$ with head $q_2(x) \leftarrow (x . :ceoOf, q_1, (y, r . :NatComp))$ from Example 3.2. The non-answer variable $y$ could be replaced by a Skolem function $f(x)$, which would yield two GAV mappings, namely $m_1 = q_2(x) \leadsto q_2(x)$ and $m_2 = q_1(x) \leadsto q_2(x)$, with respective head $q_2(x) \leftarrow (x . :ceoOf, f(x))$ and $q_2(x) \leftarrow (f(x), r . :NatComp)$. Note that Skolem functions would have to produce syntactically correct RDF values in a materialization scenario. Still in a materialization scenario, query answering would require some post-processing to prevent the values built by the Skolem functions to be accepted as answers, while in a query rewriting scenario functional values would also have to be dealt with in a special way, which in particular prevents to use off-the-shelf view-based query rewriting algorithms. Hence, value invention would be simulated here at the price of technically more complex mappings and processing. Second, the break-up of GLAV mappings into several GAV mappings would lead to higher conceptual simplicity since intrinsically connected triples, as those associated with $(x . :ceoOf, y)$ and $(y, r . :NatComp)$ in the example, could not be exposed together by a single mapping. Last but least, query rewriting would be considerably slowed down and would produce highly redundant rewritings, as demonstrated in the seminal paper [42].

Our mapping saturation (Definition 4.8) is inspired by a query saturation technique introduced in [25] to compute least general generalizations of BGPQs under RDFS background knowledge.

---

$^2$http://obdasystems.com/mastro/

$^6$https://ontop.inf.unibz.it/
It can be seen as a generalization to GLAV mappings of the T-mapping technique introduced in [43] (and further developed in [44]) to optimize query rewriting in a classical OBDA context. The T-mapping technique consists of completing the original set of GAV mappings with new ones, encapsulating information inferred from the DL ontology. For instance, given a GAV mapping \( m = q_1(x) \rightarrow q_2(x) \leftarrow C(x) \) with a class and a DL constraint specifying that \( C \) is a subclass of \( D \), a new mapping \( m' = q_1(x) \rightarrow q'_2(x) \leftarrow D(x) \) is created by composing \( m \) and the DL constraint. On this example, we would saturate the head of \( m \) into \( q_2(x) \leftarrow C(x) \land D(x) \), which is semantically equivalent to adding the mapping \( m' \). However, when mappings are GLAV and not GAV, one cannot simply add new mappings. For instance, consider the GLAV mapping \( m_1 \rightarrow q_1(x) \rightarrow q_2(x) \) with head \( q_2(x) \leftarrow (x, \text{CEOOf}, y) \). Given the entailment rule \( \text{rdfs9} \) and the ontological triple \( (\text{NatComp}, \lhd \text{sc}, \cdot \text{Comp}) \), the saturation adds the triple \( (y, \text{Comp}, \text{Comp}) \) to the body of \( q_2 \); creating instead a new mapping of the form \( m_2 = q_1(x) \rightarrow q'_2(x) \) with head \( q'_2(x) \leftarrow (y, \text{Comp}, \text{Comp}) \) would be unsatisfactory as \( q_2 \) in \( m_1 \) should correspond to the same object as \( y \) in \( m_2 \).

Our mapping saturation technique could be extended to more general entailment rules, in which the head of the rules may not be GAV, one cannot simply add new mappings. For instance, the mapping saturation technique consists of completing the original set of GAV mappings with new ones, encapsulating information inferred from the DL ontology. For instance, given a GAV mapping \( m = q_1(x) \rightarrow q_2(x) \leftarrow C(x) \) with a class and a DL constraint specifying that \( C \) is a subclass of \( D \), a new mapping \( m' = q_1(x) \rightarrow q'_2(x) \leftarrow D(x) \) is created by composing \( m \) and the DL constraint. On this example, we would saturate the head of \( m \) into \( q_2(x) \leftarrow C(x) \land D(x) \), which is semantically equivalent to adding the mapping \( m' \). However, when mappings are GLAV and not GAV, one cannot simply add new mappings. For instance, consider the GLAV mapping \( m_1 \rightarrow q_1(x) \rightarrow q_2(x) \) with head \( q_2(x) \leftarrow (x, \text{CEOOf}, y) \). Given the entailment rule \( \text{rdfs9} \) and the ontological triple \( (\text{NatComp}, \lhd \text{sc}, \cdot \text{Comp}) \), the saturation adds the triple \( (y, \text{Comp}, \text{Comp}) \) to the body of \( q_2 \); creating instead a new mapping of the form \( m_2 = q_1(x) \rightarrow q'_2(x) \) with head \( q'_2(x) \leftarrow (y, \text{Comp}, \text{Comp}) \) would be unsatisfactory as \( q_2 \) in \( m_1 \) should correspond to the same object as \( y \) in \( m_2 \).