Answering SPARQL Queries over Databases under OWL 2 QL Entailment Regime

Roman Kontchakov[1], Martin Rezk[2], Mariano Rodríguez-Muro[3], Guohui Xiao[2], and Michael Zakharyaschev[1]

[1] Department of Computer Science and Information Systems, Birkbeck, University of London, U.K.
[2] Faculty of Computer Science, Free University of Bozen-Bolzano, Italy
[3] IBM T.J. Watson Research Center, Yorktown Heights, NY, USA

Abstract. We present an extension of the ontology-based data access platform *Ontop* that supports answering SPARQL queries under the OWL 2 QL direct semantics entailment regime for data instances stored in relational databases. On the theoretical side, we show how any input SPARQL query, OWL 2 QL ontology and R2RML mappings can be rewritten to an equivalent SQL query solely over the data. On the practical side, we present initial experimental results demonstrating that by applying the *Ontop* technologies—the tree-witness query rewriting, $\mathcal{T}$-mappings compiling R2RML mappings with ontology hierarchies, and $\mathcal{T}$-mapping optimisations using SQL expressivity and database integrity constraints—the system produces scalable SQL queries.

1 Introduction

Ontology-based data access and management (OBDA) is a popular paradigm of organising access to various types of data sources that has been developed since the mid 2000s [11,17,24]. In a nutshell, OBDA separates the user from the data sources (relational databases, triple stores, etc.) by means of an ontology which provides the user with a convenient query vocabulary, hides the structure of the data sources, and can enrich incomplete data with background knowledge. About a dozen OBDA systems have been implemented in both academia and industry; e.g., [27,30,24,4,23,15,12,8,20,22]. Most of them support conjunctive queries and the OWL 2 QL profile of OWL 2 as the ontology language (or its generalisations to existential datalog rules). Thus, the OBDA platform *Ontop* [29] was designed to query data instances stored in relational databases, with the vocabularies of the data and OWL 2 QL ontologies linked by means of global-as-view (GAV) mappings. Given a conjunctive query in the vocabulary of such an ontology, *Ontop* rewrites it to an SQL query in the vocabulary of the data, optimises the rewriting and delegates its evaluation to the database system.

One of the main aims behind the newly designed query language SPARQL 1.1—a W3C recommendation since 2013—has been to support various entailment regimes, which can be regarded as variants of OBDA. Thus, the OWL 2 direct semantics entailment regime allows SPARQL queries over OWL 2 DL ontologies and RDF graphs (which can be thought of as 3-column database tables). SPARQL queries are in many aspects more expressive than conjunctive queries as they offer more complex query constructs and can retrieve not only domain elements but also class and property names using second-order variables. (Note, however, that SPARQL 1.1 does not cover all conjunctive queries.) OWL 2 DL is also vastly superior to OWL 2 QL, but this makes query answering under the OWL 2 direct semantics entailment regime intractable (CONP-hard for data complexity). For example, the query evaluation algorithm of [19] calls an OWL 2 DL reasoner for each possible assignment to the variables in a given query, and therefore cannot cope with large data instances.

In this paper, we investigate answering SPARQL queries under a less expressive entailment regime, which corresponds to OWL 2 QL, assuming that data is stored in relational databases. It is to be noted that the W3C specification[1] of SPARQL 1.1 defines entailment regimes for the profiles of OWL 2 by restricting the general definition to the profile constructs that can be used in the queries. However, in the case of OWL 2 QL, this generic approach leads to a sub-optimal, almost trivial query language, which is essentially subsumed by the OWL 2 RL entailment regime.

The first aim of this paper is to give an optimal definition of the OWL 2 QL direct semantics entailment regime and prove that—similarly to OBDA with OWL 2 QL and conjunctive queries—answering SPARQL queries under this regime is reducible to answering queries under *simple entailment*. More precisely, in Theorem 4 we construct a rewriting $\cdot^{\dagger}$ of any given SPARQL query and ontology under the OWL 2 QL entailment regime to a SPARQL query that can be evaluated on any dataset directly.

In a typical *Ontop* scenario, data is stored in a relational database whose schema is linked to the vocabulary of the given OWL 2 QL ontology via a GAV mapping in the language R2RML. The mapping allows one to transform the relational data instance into an RDF representation, called the virtual RDF graph (which is not materialised in our scenario). The rewriting $\cdot^{\dagger}$ constructs a SPARQL query over this virtual graph.

Our second aim is to show how such a SPARQL query can be translated to an equivalent SQL query over a relational representation of the virtual RDF graph as a 3-column table (translation $\boldsymbol{\tau}$ in Theorem 7). The third aim is to show that the resulting SQL query can be unfolded, using a given R2RML mapping $\mathcal{M}$, to an SQL query over the original database ($\mathsf{tr}_{\mathcal{M}}$ in Theorem 12), which is evaluated by the database system.

$$\begin{tikzcd}
\text{SPARQL query \& ontology} \arrow[r, "\dagger"] \arrow[dr, bend right, "\text{entailment regime}"'] & \text{SPARQL query} \arrow[r, "\boldsymbol{\tau}"] \arrow[d, "\text{simple entailment}"] & \text{SQL query} \arrow[r, "\mathsf{tr}_{\mathcal{M}}"] \arrow[d, "\text{evaluation}"'] & \text{SQL query} \arrow[d, "\text{evaluation}"'] \\
& \text{virtual RDF graph} & \text{triple-database} \arrow[l, "\approx"] & \text{database} \arrow[l, "\text{mapping } \mathcal{M}"]
\end{tikzcd}$$

Unfortunately, each of these three transformations may involve an exponential blowup. We tackle this problem in *Ontop* using the following optimisation techniques. (*i*) The mapping is compiled with the ontology into a $\mathcal{T}$-mapping [29] and optimised by database dependencies (e.g., primary, candidate and foreign keys) and SQL disjunctions. (*ii*) The SPARQL-to-SQL translation is optimised using null join elimination (Theorem 8). (*iii*) The unfolding is optimised by eliminating joins with mismatching R2RML IRI templates, de-IRIing the join conditions (Section 3.3) and using database dependencies.

Our contributions (Theorems 4, 7, 8 and 12 and optimisations in Section 3.3) make *Ontop* the first system to support the W3C recommendations OWL 2 QL, R2RML, SPARQL and the OWL 2 QL direct semantics entailment regime; its architecture is out-

[1] `http://www.w3.org/TR/sparql11-entailment`

lined in Section 4. We evaluate the performance of *Ontop* using the LUBM Benchmark [16] extended with queries containing class and property variables, and compare it with two other systems that support the OWL 2 entailment regime by calling OWL DL reasoners (Section 5). Our experiments show that *Ontop* outperforms the reasoner-based systems for most of the queries over small datasets; over larger datasets the difference becomes dramatic, with *Ontop* demonstrating a solid performance even on 69 million triples in LUBM_{500}. Finally, we note that, although Ontop was designed to work with existing relational databases, it is also applicable in the context of RDF triple stores, in which case approaches such as the one from [3] can be used to generate suitable relational schemas. Omitted proofs and evaluation details can be found in the full version at `http://www.dcs.bbk.ac.uk/~michael/ISWC-14-v2.pdf`.

2 SPARQL Queries under OWL 2 QL Entailment Regime

SPARQL is a W3C standard language designed to query RDF graphs. Its vocabulary contains four pairwise disjoint and countably infinite sets of symbols: I for ***IRIs***, B for *blank nodes*, L for *RDF literals*, and V for *variables*. The elements of $\mathsf{C} = \mathsf{I} \cup \mathsf{B} \cup \mathsf{L}$ are called *RDF terms*. A *triple pattern* is an element of $(\mathsf{C} \cup \mathsf{V}) \times (\mathsf{I} \cup \mathsf{V}) \times (\mathsf{C} \cup \mathsf{V})$. A *basic graph pattern* (*BGP*) is a finite set of triple patterns. Finally, a *graph pattern*, P, is an expression defined by the grammar

$$P \ ::= \ \text{BGP} \ \mid \ \textsc{Filter}(P, F) \ \mid \ \textsc{Bind}(P, v, c) \ \mid \ \textsc{Union}(P_1, P_2) \ \mid \ \textsc{Join}(P_1, P_2) \ \mid \ \textsc{Opt}(P_1, P_2, F),$$

where F, a *filter*, is a formula constructed from atoms of the form *bound*(v), $(v = c)$, $(v = v')$, for $v, v' \in \mathsf{V}$, $c \in \mathsf{C}$, and possibly other built-in predicates using the logical connectives $\land$ and $\neg$. The set of variables in P is denoted by *var*(P).

A *SPARQL query* is a graph pattern P with a *solution modifier*, which specifies the *answer variables*—the variables in P whose values we are interested in—and the form of the output (we ignore other solution modifiers for simplicity). The values to variables are given by *solution mappings*, which are *partial* maps $s\colon \mathsf{V} \to \mathsf{C}$ with (possibly empty) domain *dom*(s). In this paper, we use the set-based (rather than bag-based, as in the specification) semantics for SPARQL. For sets S_1 and S_2 of solution mappings, a filter F, a variable $v \in \mathsf{V}$ and a term $c \in \mathsf{C}$, let

- $\textsc{Filter}(S, F) = \{s \in S \mid F^s = \top\}$;
- $\textsc{Bind}(S, v, c) = \{s \oplus \{v \mapsto c\} \mid s \in S\}$ (provided that $v \notin dom(s)$, for $s \in S$);
- $\textsc{Union}(S_1, S_2) = \{s \mid s \in S_1$ or $s \in S_2\}$;
- $\textsc{Join}(S_1, S_2) = \{s_1 \oplus s_2 \mid s_1 \in S_1$ and $s_2 \in S_2$ are compatible$\}$;
- $\textsc{Opt}(S_1, S_2, F) = \textsc{Filter}(\textsc{Join}(S_1, S_2), F) \ \cup \ \{s_1 \in S_1 \mid$ for all $s_2 \in S_2$, either s_1, s_2 are incompatible or $F^{s_1 \oplus s_2} \neq \top\}$.

Here, s_1 and s_2 are *compatible* if $s_1(v) = s_2(v)$, for any $v \in dom(s_1) \cap dom(s_2)$, in which case $s_1 \oplus s_2$ is a solution mapping with $s_1 \oplus s_2\colon v \mapsto s_1(v)$, for $v \in dom(s_1)$, $s_1 \oplus s_2\colon v \mapsto s_2(v)$, for $v \in dom(s_2)$, and domain $dom(s_1) \cup dom(s_2)$. The *truth-value* $F^s \in \{\top, \bot, \varepsilon\}$ *of a filter* F *under a solution mapping* s is defined inductively:

- $(\mathit{bound}(v))^s$ is $\top$ if $v \in \mathit{dom}(s)$ and $\bot$ otherwise;
- $(v = c)^s = \varepsilon$ if $v \notin \mathit{dom}(s)$; otherwise, $(v = c)^s$ is the classical truth-value of the predicate $s(v) = c$; similarly, $(v = v')^s = \varepsilon$ if either v or $v' \notin \mathit{dom}(s)$; otherwise, $(v = v')^s$ is the classical truth-value of the predicate $s(v) = s(v')$;
- $(\neg F)^s = \begin{cases} \varepsilon, & \text{if } F^s = \varepsilon, \\ \neg F^s, & \text{otherwise,} \end{cases}$ and $(F_1 \land F_2)^s = \begin{cases} \bot, & \text{if } F_1^s = \bot \text{ or } F_2^s = \bot, \\ \top, & \text{if } F_1^s = F_2^s = \top, \\ \varepsilon, & \text{otherwise.} \end{cases}$

Finally, given an RDF graph G, the *answer to a graph pattern P over G* is the set $[\![P]\!]_G$ of solution mappings defined by induction using the operations above and starting from the following base case: for a basic graph pattern B,

$$[\![B]\!]_G = \{s \colon \mathit{var}(B) \to \mathsf{C} \mid s(B) \subseteq G\}, \tag{1}$$

where $s(B)$ is the set of triples resulting from substituting each variable u in B by $s(u)$. This semantics is known as *simple entailment*.

Remark 1. The condition '$F^{s_1 \oplus s_2}$ *is not true*' in the definition of OPT is different from '$F^{s_1 \oplus s_2}$ has an effective Boolean value of false' given by the W3C specification:[2] the effective Boolean value can be undefined (type error) if a variable in F is not bound by $s_1 \oplus s_2$. As we shall see in Section 3.1, our reading corresponds to LEFT JOIN in SQL. (Note also that the informal explanation of OPT in the W3C specification is inconsistent with the definition of DIFF; see the full version for details.)

Under the *OWL 2 QL direct semantics entailment regime*, one can query an RDF graph G that consist of two parts: an *extensional* sub-graph $\mathcal{A}$ representing the *data* as OWL 2 QL class and property assertions, and the *intensional* sub-graph $\mathcal{T}$ representing the background *knowledge* as OWL 2 QL class and property axioms. We write $(\mathcal{T}, \mathcal{A})$ in place of G to emphasise the partitioning. To illustrate, we give a simple example.

Example 2. Consider the following two axioms from the LUBM ontology $(\mathcal{T}, \mathcal{A})$ (see Section 5), which are given here in the functional-style syntax (FSS):

SubClassOf(ub:UGStudent, ub:Student), SubClassOf(ub:GradStudent, ub:Student).

Under the entailment regime, we can write a query that retrieves all named *subclasses* of students in $(\mathcal{T}, \mathcal{A})$ and all *instances* of each of these subclasses (cf. q_9' in Section 5):

SELECT $?x$ $?C$ WHERE { $?C$ rdfs:subClassOf ub:Student. $?x$ rdf:type $?C$. }.

Here $?C$ ranges over the class names (IRIs) in $(\mathcal{T}, \mathcal{A})$ and $?x$ over the IRIs of individuals. If, for example, $\mathcal{A}$ consists of the two assertions on the left-hand side, then the answer to the query over $(\mathcal{T}, \mathcal{A})$ is on the right-hand side:

$\mathcal{A}$
ClassAssertion(ub:UGStudent, ub:jim)
ClassAssertion(ub:Student, ub:bob)

$?x$	$?C$
ub:jim	ub:UGStudent
ub:jim	ub:Student
ub:bob	ub:Student

[2] http://www.w3.org/TR/sparql11-query/#sparqlAlgebra

To formally define SPARQL queries that can be used under the OWL 2 QL direct semantics entailment regime, we assume that the set I of IRIs is partitioned into disjoint and countably infinite sets of *class names* I_C, *object property names* I_R and *individual names* I_I. Similarly, the variables V are also assumed to be a disjoint union of countably infinite sets V_C, V_R, V_I. Now, we define an *OWL 2 QL BGP* as a finite set of triple patterns representing OWL 2 QL axiom and assertion templates in the FSS such as:[3]

$$\begin{array}{ll}
\mathsf{SubClassOf}(\mathit{SubC}, \mathit{SuperC}), & \mathsf{DisjointClasses}(\mathit{SubC}_1, \dots, \mathit{SubC}_n),\\
\mathsf{ObjectPropertyDomain}(\mathit{OP}, \mathit{SuperC}), & \mathsf{ObjectPropertyRange}(\mathit{OP}, \mathit{SuperC}),\\
\mathsf{SubObjectPropertyOf}(\mathit{OP}, \mathit{OP}), & \mathsf{DisjointObjectProperties}(\mathit{OP}_1, \dots, \mathit{OP}_n),\\
\mathsf{ClassAssertion}(\mathit{SuperC}, \mathit{I}), & \mathsf{ObjectPropertyAssertion}(\mathit{OP}, \mathit{I}, \mathit{I}),
\end{array}$$

where $I \in \mathsf{I}_I \cup \mathsf{V}_I$ and *OP*, *SubC* and *SuperC* are defined by the following grammar with $C \in \mathsf{I}_C \cup \mathsf{V}_C$ and $R \in \mathsf{I}_R \cup \mathsf{V}_R$:

$$\begin{array}{rcll}
\mathit{OP} & ::= & R & \mid\ \mathsf{ObjectInverseOf}(R),\\
\mathit{SubC} & ::= & C & \mid\ \mathsf{ObjectSomeValuesFrom}(\mathit{OP}, \mathsf{owl{:}Thing}),\\
\mathit{SuperC} & ::= & C & \mid\ \mathsf{ObjectIntersectionOf}(\mathit{SuperC}_1, \dots, \mathit{SuperC}_n) \ \mid\\
& & & \qquad\qquad \mathsf{ObjectSomeValuesFrom}(\mathit{OP}, \mathit{SuperC}).
\end{array}$$

OWL 2 QL graph patterns are constructed from OWL 2 QL BGPs using the SPARQL operators. Finally, an *OWL 2 QL query* is a pair (P, V), where P is an OWL 2 QL graph pattern and $V \subseteq \mathit{var}(P)$. To define the answer to such a query (P, V) over an RDF graph $(\mathcal{T}, \mathcal{A})$, we fix a *finite* vocabulary $\mathsf{I}_{\mathcal{T},\mathcal{A}} \subseteq \mathsf{I}$ that includes all names (IRIs) in $\mathcal{T}$ and $\mathcal{A}$ as well as the required finite part of the OWL 2 RDF-based vocabulary (e.g., owl:Thing but not the infinite number of the rdf:_n). To ensure finiteness of the answers and proper typing of variables, in the following definition we only consider solution mappings $s\colon \mathit{var}(P) \to \mathsf{I}_{\mathcal{T},\mathcal{A}}$ such that $s^{-1}(\mathsf{I}_\alpha) \subseteq \mathsf{V}_\alpha$, for $\alpha \in \{C, R, I\}$. For each BGP B, we define the *answer* $[\![B]\!]_{\mathcal{T},\mathcal{A}}$ *to* B *over* $(\mathcal{T}, \mathcal{A})$ by taking

$$[\![B]\!]_{\mathcal{T},\mathcal{A}} \ = \ \{s\colon \mathit{var}(B) \to \mathsf{I}_{\mathcal{T},\mathcal{A}} \mid (\mathcal{T}, \mathcal{A}) \models s(B)\},$$

where $\models$ is the entailment relation given by the OWL 2 direct semantics. Starting from the $[\![B]\!]_{\mathcal{T},\mathcal{A}}$ and applying the SPARQL operators in P, we compute the set $[\![P]\!]_{\mathcal{T},\mathcal{A}}$ of *solution mappings*. The *answer to* (P, V) *over* $(\mathcal{T}, \mathcal{A})$ is the restriction $[\![P]\!]_{\mathcal{T},\mathcal{A}}|_V$ of the solution mappings in $[\![P]\!]_{\mathcal{T},\mathcal{A}}$ to the variables in V.

Example 3. Suppose $\mathcal{T}$ contains

$$\begin{array}{l}
\mathsf{SubClassOf}(\mathsf{{:}A}, \mathsf{ObjectSomeValuesFrom}(\mathsf{{:}P}, \mathsf{owl{:}Thing})),\\
\mathsf{SubObjectPropertyOf}(\mathsf{{:}P}, \mathsf{{:}R}), \quad \mathsf{SubObjectPropertyOf}(\mathsf{{:}P}, \mathsf{ObjectInverseOf}(\mathsf{{:}S})).
\end{array}$$

Consider the following OWL 2 QL BGP B:

$$\begin{array}{l}
\mathsf{ClassAssertion}(\mathsf{ObjectSomeValuesFrom}(\mathsf{{:}R}, \mathsf{ObjectSomeValuesFrom}(\mathsf{{:}S},\\
\qquad\qquad\qquad\qquad \mathsf{ObjectSomeValuesFrom}(\mathsf{{:}T}, \mathsf{owl{:}Thing}))), ?x).
\end{array}$$

---

[3] The official specification of legal queries under the OWL 2 QL entailment regime only allows $\mathsf{ClassAssertion}(C, I)$ rather than $\mathsf{ClassAssertion}(\mathit{SuperC}, I)$, which makes the OWL 2 QL entailment regime trivial and essentially subsumed by the OWL 2 RL entailment regime.

Assuming that $\mathcal{A} = \{\mathsf{ClassAssertion}(:\mathsf{A}, :\mathsf{a}), \mathsf{ObjectPropertyAssertion}(:\mathsf{T}, :\mathsf{a}, :\mathsf{b})\}$, it is not hard to see that $[\![B]\!]_{\mathcal{T},\mathcal{A}} = \{?x \mapsto :\mathsf{a}\}$. Indeed, by the first assertion of $\mathcal{A}$ and the first two axioms of $\mathcal{T}$, any model of $(\mathcal{T}, \mathcal{A})$ contains a domain element w (not necessarily among the individuals in $\mathcal{A}$) such that $\mathsf{ObjectPropertyAssertion}(:\mathsf{R}, :\mathsf{a}, w)$ holds. In addition, the third axiom of $\mathcal{T}$ implies $\mathsf{ObjectPropertyAssertion}(:\mathsf{S}, w, :\mathsf{a})$, which together with the second assertion of $\mathcal{A}$ mean that $\{?x \mapsto :\mathsf{a}\}$ is an answer.

The following theorem shows that answering OWL 2 QL queries under the direct semantics entailment regime can be reduced to answering OWL 2 QL queries under simple entailment, which are evaluated only on the extensional part of the RDF graph:

Theorem 4. *Given any intensional graph $\mathcal{T}$ and OWL 2 QL query (P, V), one can construct an OWL 2 QL query $(P^\dagger, V)$ such that, for any extensional graph $\mathcal{A}$ (in some fixed finite vocabulary), $[\![P]\!]_{\mathcal{T},\mathcal{A}}|_V \;=\; [\![P^\dagger]\!]_{\mathcal{A}}|_V$.*

Proof sketch. By the definition of the entailment regime, it suffices to construct $B^\dagger$, for any *BGP* B; the rewriting $P^\dagger$ is obtained then by replacing each BGP B in P with $B^\dagger$. First, we instantiate the class and property variables in B by all possible class and property names in the given vocabulary and add the respective BIND operations. In each of the resulting BGPs, we remove the class and property axioms if they are entailed by $\mathcal{T}$; otherwise we replace the BGP with an empty one. The obtained BGPs are (SPARQL representations of) conjunctive queries (with non-distinguished variables in complex concepts *SuperC* of the assertions $\mathsf{ClassAssertion}(\textit{SuperC}, \textit{I})$). The second step is to rewrite these conjunctive queries together with $\mathcal{T}$ into unions of conjunctive queries (BGPs) that can be evaluated over any extensional graph $\mathcal{A}$ [5,21]. (We emphasise that the SPARQL algebra operations, including difference and OPT, are applied to BGPs and do not interact with the two steps of our rewriting.) ❑

We illustrate the proof of Theorem 4 using the queries from Examples 2 and 3.

Example 5. The class variable $?C$ in the query from Example 2 can be instantiated, using BIND, by all possible values from $\mathsf{I}_C \cap \mathsf{I}_{\mathcal{T},\mathcal{A}}$, which gives the rewriting

```
SELECT ?x ?C WHERE {
    { ?x rdf:type ub:Student. BIND(ub:Student as ?C) } UNION
    { ?x rdf:type ub:GradStudent. BIND(ub:GradStudent as ?C) } UNION
    { ?x rdf:type ub:UGStudent. BIND(ub:UGStudent as ?C) } }.
```

The query from Example 3 is equivalent to a (tree-shaped) conjunctive query with three non-distinguished and one answer variable, which can be rewritten to

```
SELECT ?x WHERE { { ?x :R ?y. ?y :S ?z. ?z :T ?u. } UNION
                  { ?x rdf:type :A. ?x :T ?u. } }.
```

3 Translating SPARQL under Simple Entailment to SQL

A number of translations of SPARQL queries (under simple entailment) to SQL queries have already been suggested in the literature; see, e.g., [9,13,7,32,27]. However, none of them is suitable for our aims because they do not take into account the three-valued logic used in the OPTIONAL and BOUND constructs of the current SPARQL 1.1 (the semantics of OPTIONAL was not compositional in SPARQL 1.0). Note also that SPARQL has been translated to Datalog [25,2,26].

We begin by recapping the basics of relational algebra and SQL (see e.g., [1]). Let U be a finite (possibly empty) set of *attributes*. A *tuple over* U is a map $t\colon U \to \Delta$, where Δ is the underlying domain, which always contains a distinguished element *null*. A ($|U|$*-ary*) *relation over* U is a finite set of tuples over U (again, we use the set-based rather than bag-based semantics). A *filter* F *over* U is a formula constructed from atoms *isNull*(U'), $(u = c)$ and $(u = u')$, where $U' \subseteq U$, $u, u' \in U$ and $c \in \Delta$, using the connectives $\land$ and $\neg$. Let F be a filter with variables U and let t be a tuple over U. The *truth-value* $F^t \in \{\top, \bot, \varepsilon\}$ *of* F *over* t is defined inductively:

- $(\mathit{isNull}(U'))^t$ is $\top$ if $t(u)$ is *null*, for all $u \in U'$, and $\bot$ otherwise;
- $(u = c)^t = \varepsilon$ if $t(u)$ is *null*; otherwise, $(u = c)^t$ is the classical truth-value of the predicate $t(u) = c$; similarly, $(u = u')^t = \varepsilon$ if either $t(u)$ or $t(u')$ is *null*; otherwise, $(u = u')^t$ is the classical truth-value of the predicate $t(u) = t(u')$;
- $(\neg F)^t = \begin{cases} \varepsilon, & \text{if } F^t = \varepsilon, \\ \neg F^t, & \text{otherwise,} \end{cases}$ and $(F_1 \land F_2)^t = \begin{cases} \bot, & \text{if } F_1^t = \bot \text{ or } F_2^t = \bot, \\ \top, & \text{if } F_1^t = F_2^t = \top, \\ \varepsilon, & \text{otherwise.} \end{cases}$

(Note that $\neg$ and $\land$ are interpreted in the same three-valued logic as in SPARQL.) We use standard relational algebra operations such as union, difference, projection, selection, renaming and natural (inner) join. Let R_i be a relation over U_i, $i = 1, 2$.

- If $U_1 = U_2$ then the standard $R_1 \cup R_2$ and $R_1 \setminus R_2$ are relations over U_1.
- If $U \subseteq U_1$ then $\pi_U R_1 = R_1|_U$ is a relation over U.
- If F is a filter over U_1 then $\sigma_F R_1 = \{t \in R_1 \mid F^t = \top\}$ is a relation over U_1.
- If $v \notin U_1$ and $u \in U_1$ then $\rho_{v/u} R_1 = \{t_{v/u} \mid t \in R_1\}$, where $t_{v/u}\colon v \mapsto t(u)$ and $t_{v/u}\colon u' \mapsto t(u')$, for $u' \in U_1 \setminus \{u\}$, is a relation over $(U_1 \setminus \{u\}) \cup \{v\}$.
- $R_1 \bowtie R_2 = \{t_1 \oplus t_2 \mid t_1 \in R_1 \text{ and } t_2 \in R_2 \text{ are compatible}\}$ is a relation over $U_1 \cup U_2$. Here, t_1 and t_2 are *compatible* if $t_1(u) = t_2(u) \neq \mathit{null}$, for all $u \in U_1 \cap U_2$, in which case a tuple $t_1 \oplus t_2$ over $U_1 \cup U_2$ is defined by taking $t_1 \oplus t_2\colon u \mapsto t_1(u)$, for $u \in U_1$, and $t_1 \oplus t_2\colon u \mapsto t_2(u)$, for $u \in U_2$ (note that if u is *null* in either of the tuples then they are incompatible).

To bridge the gap between partial functions (solution mappings) in SPARQL and total mappings (on attributes) in SQL, we require one more operation (expressible in SQL):

- If $U \cap U_1 = \emptyset$ then the *padding* $\mu_U R_1$ is $R_1 \bowtie \mathit{null}^U$, where null^U is the relation consisting of a single tuple t over U with $t\colon u \mapsto \mathit{null}$, for all $u \in U$.

By an *SQL query*, Q, we understand any expression constructed from relation symbols (each over a fixed set of attributes) and filters using the relational algebra operations given above (and complying with all restrictions on the structure). Suppose Q is an SQL query and D a data instance which, for any relation symbol in the schema under consideration, gives a concrete relation over the corresponding set of attributes. The *answer to* Q *over* D is a relation $\|Q\|_D$ defined inductively in the obvious way starting from the base case: for a relation symbol Q, $\|Q\|_D$ is the corresponding relation in D.

We now define a translation, $\boldsymbol{\tau}$, which, given a graph pattern P, returns an SQL query $\boldsymbol{\tau}(P)$ with the same answers as P. More formally, for a set of variables V, let ext_V be a function transforming any solution mapping s with $\mathit{dom}(s) \subseteq V$ to a tuple over V by padding it with *null*s:

$$\mathit{ext}_V(s) \;=\; \{v \mapsto s(v) \mid v \in \mathit{dom}(s)\} \;\cup\; \{v \mapsto \mathit{null} \mid v \in V \setminus \mathit{dom}(s)\}.$$

The *relational answer to* P *over* G is $\|P\|_G = \{\mathit{ext}_{\mathit{var}(P)}(s) \mid s \in [\![P]\!]_G\}$. The SQL query $\boldsymbol{\tau}(P)$ will be such that, for any RDF graph G, the relational answer to P over G coincides with the answer to $\boldsymbol{\tau}(P)$ over $\mathit{triple}(G)$, the database instance storing G as a ternary relation *triple* with the attributes *subj*, *pred*, *obj*. First, we define the translation of a SPARQL filter F by taking $\boldsymbol{\tau}(F)$ to be the SQL filter obtained by replacing each $\mathit{bound}(v)$ with $\neg \mathit{isNull}(v)$ (other built-in predicates can be handled similarly).

Proposition 6. *Let* F *be a SPARQL filter and let* V *be the set of variables in* F. *Then* $F^s = (\boldsymbol{\tau}(F))^{\mathit{ext}_V(s)}$, *for any solution mapping* s *with* $\mathit{dom}(s) \subseteq V$.

The definition of $\boldsymbol{\tau}$ proceeds by induction on the construction of P. Note that we can always assume that graph patterns *under simple entailment* do not contain blank nodes because they can be replaced by fresh variables. It follows that a BGP $\{\mathit{tp}_1, \dots, \mathit{tp}_n\}$ is equivalent to $\textsc{Join}(\{\mathit{tp}_1\}, \textsc{Join}(\{\mathit{tp}_2\}, \dots))$. So, for the basis of induction we set

$$\boldsymbol{\tau}(\{\langle s,p,o\rangle\}) = \begin{cases} \pi_\emptyset \sigma_{(\mathit{subj}=s)\wedge(\mathit{pred}=p)\wedge(\mathit{obj}=o)}\, \mathit{triple}, & \text{if } s,p,o \in \mathsf{I} \cup \mathsf{L},\\ \pi_s \rho_{s/\mathit{subj}}\, \sigma_{(\mathit{pred}=p)\wedge(\mathit{obj}=o)}\, \mathit{triple}, & \text{if } s \in \mathsf{V} \text{ and } p,o \in \mathsf{I} \cup \mathsf{L},\\ \pi_{s,o} \rho_{s/\mathit{subj}}\, \rho_{o/\mathit{obj}}\, \sigma_{\mathit{pred}=p}\, \mathit{triple}, & \text{if } s,o \in \mathsf{V}, s \neq o, p \in \mathsf{I} \cup \mathsf{L},\\ \pi_s \rho_{s/\mathit{subj}}\, \sigma_{(\mathit{pred}=p)\wedge(\mathit{subj}=\mathit{obj})}\, \mathit{triple}, & \text{if } s,o \in \mathsf{V}, s = o, p \in \mathsf{I} \cup \mathsf{L},\\ \dots \end{cases}$$

(the remaining cases are similar). Now, if P_1 and P_2 are graph patterns and F_1 and F are filters containing only variables in $\mathit{var}(P_1)$ and $\mathit{var}(P_1) \cup \mathit{var}(P_2)$, respectively, then we set $U_i = \mathit{var}(P_i)$, $i = 1, 2$, and

$$\begin{aligned}
&\boldsymbol{\tau}(\textsc{Filter}(P_1, F_1)) = \sigma_{\boldsymbol{\tau}(F_1)} \boldsymbol{\tau}(P_1),\\
&\boldsymbol{\tau}(\textsc{Bind}(P_1, v, c)) = \boldsymbol{\tau}(P_1) \bowtie \{v \mapsto c\},\\
&\boldsymbol{\tau}(\textsc{Union}(P_1, P_2)) = \;\mu_{U_2 \setminus U_1} \boldsymbol{\tau}(P_1) \;\cup\; \mu_{U_1 \setminus U_2} \boldsymbol{\tau}(P_2),\\
&\boldsymbol{\tau}(\textsc{Join}(P_1, P_2)) = \bigcup_{\substack{V_1, V_2 \subseteq U_1 \cap U_2\\ V_1 \cap V_2 = \emptyset}} \mu_{V_1 \cup V_2} \big[(\pi_{U_1 \setminus V_1} \sigma_{\mathit{isNull}(V_1)} \boldsymbol{\tau}(P_1)) \bowtie (\pi_{U_2 \setminus V_2} \sigma_{\mathit{isNull}(V_2)} \boldsymbol{\tau}(P_2))\big],\\
&\boldsymbol{\tau}(\textsc{Opt}(P_1, P_2, F)) = \;\; \sigma_{\boldsymbol{\tau}(F)}(\boldsymbol{\tau}(\textsc{Join}(P_1, P_2))) \;\cup\\
&\qquad\qquad\qquad\qquad\qquad \mu_{U_2 \setminus U_1} \big(\boldsymbol{\tau}(P_1) \setminus \pi_{U_1} \sigma_{\boldsymbol{\tau}(F)}(\boldsymbol{\tau}(\textsc{Join}(P_1, P_2)))\big).
\end{aligned}$$

It is readily seen that any $\boldsymbol{\tau}(P)$ is a valid SQL query and defines a relation over $\mathit{var}(P)$.

Theorem 7. *For any RDF graph* G *and any graph pattern* P, $\|P\|_G = \|\boldsymbol{\tau}(P)\|_{\mathit{triple}(G)}$.

Proof. The proof is by induction on the structure of P. Here we only consider the induction step for $P = \textsc{Join}(P_1, P_2)$. Let $U_i = \mathit{var}(P_i)$, $i = 1, 2$, and $U = U_1 \cap U_2$.

If $t \in \|\textsc{Join}(P_1, P_2)\|_G$ then there is a solution mapping $s \in [\![\textsc{Join}(P_1, P_2)]\!]_G$ with $\mathit{ext}_{U_1 \cup U_2}(s) = t$, and so there are $s_i \in [\![P_i]\!]_G$ such that s_1 and s_2 are compatible and $s_1 \oplus s_2 = s$. Since, $\mathit{ext}_{U_i}(s_i) \in \|P_i\|_G$, by IH, $\mathit{ext}_{U_i}(s_i) \in \|\boldsymbol{\tau}(P_i)\|_{\mathit{triple}(G)}$. Let $V = \mathit{dom}(s_1) \cap \mathit{dom}(s_2)$ and $V_i = U \setminus \mathit{dom}(s_i)$. Then V_1, V_2 and V are disjoint and partition U. By definition, $\mathit{ext}_{U_i}(s_i)\colon v \mapsto \mathit{null}$, for each $v \in V_i$, and therefore $\mathit{ext}_{U_i}(s_i)$ is in $\|\sigma_{\mathit{isNull}(V_i)}\boldsymbol{\tau}(P_i)\|_{\mathit{triple}(G)}$. Let $t_i = \mathit{ext}_{U_i \setminus V_i}(s_i)$ and $Q_i = \pi_{U_i \setminus V_i}(\sigma_{\mathit{isNull}(V_i)}\boldsymbol{\tau}(P_i))$. We have $t_i \in \|Q_i\|_{\mathit{triple}(G)}$, and since s_1 and s_2 are compatible and V are the common non-null attributes of t_1 and t_2, we obtain $t_1 \oplus t_2 \in \|Q_1 \bowtie Q_2\|_{\mathit{triple}(G)}$. As t extends $t_1 \oplus t_2$ to $V_1 \cup V_2$ by *null*s, we have $t \in \|\boldsymbol{\tau}(\textsc{Join}(P_1, P_2))\|_{\mathit{triple}(G)}$.

If $t \in \|\boldsymbol{\tau}(\textsc{Join}(P_1, P_2))\|_{\mathit{triple}(G)}$ then there are disjoint $V_1, V_2 \subseteq U$ and compatible tuples t_1 and t_2 such that $t_i \in \|\pi_{U_i \setminus V_i}(\sigma_{\mathit{isNull}(V_i)}\boldsymbol{\tau}(P_i))\|_{\mathit{triple}(G)}$ and t extends $t_1 \oplus t_2$ to $V_1 \cup V_2$ by *null*s. Let $s_i = \{v \mapsto t(v) \mid v \in U_i \text{ and } t(v) \text{ is not } \mathit{null}\}$. Then s_1 and s_2 are compatible and $\mathit{ext}_{U_i}(s_i) \in \|\boldsymbol{\tau}(P_i)\|_{\mathit{triple}(G)}$. By IH, $\mathit{ext}_{U_i}(s_i) \in \|P_i\|_G$ and $s_i \in [\![P_i]\!]_G$. So, $s_1 \oplus s_2 \in [\![\textsc{Join}(P_1, P_2)]\!]_G$ and $\mathit{ext}_{U_1 \cup U_2}(s_1 \oplus s_2) = t \in \|\textsc{Join}(P_1, P_2)\|_G$. ❑

3.1 Optimising SPARQL Join and Opt

By definition, $\boldsymbol{\tau}(\textsc{Join}(P_1, P_2))$ is a union of exponentially many natural joins ($\bowtie$). Observe, however, that for any BGP $B = \{\mathit{tp}_1, \ldots, \mathit{tp}_n\}$, none of the attributes in the $\boldsymbol{\tau}(\mathit{tp}_i)$ can be *null*. So, we can drastically simplify the definition of $\boldsymbol{\tau}(B)$ by taking

$$\boldsymbol{\tau}(\{\mathit{tp}_1, \ldots, \mathit{tp}_n\}) = \boldsymbol{\tau}(\mathit{tp}_1) \bowtie \cdots \bowtie \boldsymbol{\tau}(\mathit{tp}_n).$$

Moreover, this observation can be generalised. First, we identify the variables in graph patterns that are not necessarily bound in solution mappings:

$$\begin{aligned}
\nu(B) &= \emptyset, \qquad B \text{ is a BGP},\\
\nu(\textsc{Filter}(P_1, F)) &= \nu(P_1) \setminus \{v \mid \mathit{bound}(v) \text{ is a conjunct of } F\},\\
\nu(\textsc{Bind}(P_1, v, c)) &= \nu(P_1),\\
\nu(\textsc{Union}(P_1, P_2)) &= (\mathit{var}(P_1) \setminus \mathit{var}(P_2)) \cup (\mathit{var}(P_2) \setminus \mathit{var}(P_1)) \cup \nu(P_1) \cup \nu(P_2),\\
\nu(\textsc{Join}(P_1, P_2)) &= \nu(P_1) \cup \nu(P_2),\\
\nu(\textsc{Opt}(P_1, P_2, F)) &= \nu(P_1) \cup \mathit{var}(P_2).
\end{aligned}$$

Thus, if a variable v in P does not belong to $\nu(P)$, then $v \in \mathit{dom}(s)$, for any solution mapping $s \in [\![P]\!]_G$ and RDF graph G (but not the other way round). Now, we observe that the union in the definition of $\boldsymbol{\tau}(\textsc{Join}(P_1, P_2))$ can be taken over those subsets of $\mathit{var}(P_1) \cap \mathit{var}(P_2)$ that only contain variables from $\nu(P_1) \cup \nu(P_2)$. This gives us:

Theorem 8. *If $\mathit{var}(P_1) \cap \mathit{var}(P_2) \cap (\nu(P_1) \cup \nu(P_2)) = \emptyset$ then we can define*

$$\boldsymbol{\tau}(\textsc{Join}(P_1, P_2)) = \boldsymbol{\tau}(P_1) \bowtie \boldsymbol{\tau}(P_2), \quad \boldsymbol{\tau}(\textsc{Opt}(P_1, P_2, F)) = \boldsymbol{\tau}(P_1) ⟕_{\boldsymbol{\tau}(F)} \boldsymbol{\tau}(P_2),$$

where $R_1 ⟕_F R_2 = \sigma_F(R_1 \bowtie R_2) \cup \mu_{U_2 \setminus U_1}(R_1 \setminus \pi_{U_1}(\sigma_F(R_1 \bowtie R_2)))$, for R_i over U_i.

(Note that the relational operation $⟕_F$ corresponds to LEFT JOIN in SQL with the condition F placed in its ON clause.)

Example 9. Consider the following BGP B taken from the official SPARQL specification ('find the names of people who do not know anyone'):

$$\textsc{Filter}(\textsc{Opt}(\{\,?x \ \text{foaf:givenName} \ ?n\,\},\ \{\,?x \ \text{foaf:knows} \ ?w\,\},\ \top), \neg \mathit{bound}(?w)).$$

By Theorem 8, $\boldsymbol{\tau}(B)$ is defined as $\sigma_{\mathit{isNull}(w)}(\pi_{x,n}Q_1 ⟕ \pi_{x,w}Q_2)$, where Q_1 and Q_2 are $\sigma_{\mathit{pred}=\text{foaf:givenName}}\rho_{x/\mathit{subj}}\rho_{n/\mathit{obj}}\ \mathit{triple}$ and $\sigma_{\mathit{pred}=\text{foaf:knows}}\rho_{x/\mathit{subj}}\rho_{w/\mathit{obj}}\ \mathit{triple}$, respectively (we note in passing that the projection on x is equivalent to $\pi_x Q_1 \setminus \pi_x Q_2$).

3.2 R2RML Mappings

The SQL translation of a SPARQL query constructed above has to be evaluated over the ternary relation *triple*(G) representing the virtual RDF graph G. Our aim now is to transform it to an SQL query over the actual database, which is related to G by means of an R2RML mapping [10]. A variant of such a transformation has been suggested in [27]. Here we develop the idea first presented in [28]. We begin with a simple example.

Example 10. The following R2RML mapping (in the Turtle syntax) populates an object property ub:UGDegreeFrom from a relational table students, whose attributes id and degreeuniid identify graduate students and their universities:

```
_:m1 a rr:TripleMap;
      rr:logicalTable [ rr:sqlQuery "SELECT * FROM students WHERE stype=1" ];
      rr:subjectMap [ rr:template "/GradStudent{id}" ] ;
      rr:predicateObjectMap [ rr:predicate ub:UGDegreeFrom ;
                              rr:objectMap [ rr:template "/Uni{degreeuniid}" ] ]
```

More specifically, for each tuple in the query, an R2RML processor generates an RDF triple with the predicate ub:UGDegreeFrom and the subject and object constructed from attributes id and degreeuniid, respectively, using IRI templates.

Our aim now is as follows: given an R2RML mapping $\mathcal{M}$, we are going to define an SQL query $\mathsf{tr}_{\mathcal{M}}(\mathit{triple})$ that constructs the relational representation $\mathit{triple}(G_{D,\mathcal{M}})$ of the virtual RDF graph $G_{D,\mathcal{M}}$ obtained by $\mathcal{M}$ from any given data instance D. Without loss of generality and to simplify presentation, we assume that each triple map has

- one logical table (rr:sqlQuery),
- one subject map (rr:subjectMap), which does not have resource typing (rr:class),
- and one predicate-object map with one rr:predicateMap and one rr:objectMap.

This normal form can be achieved by introducing predicate-object maps with rdf:type and splitting any triple map into a number of triple maps with the same logical table and subject. We also assume that triple maps contain no referencing object maps (rr:parentTriplesMap, etc.) since they can be eliminated using joint SQL queries [10]. Finally, we assume that the term maps (i.e., subject, predicate and object maps) contain no constant shortcuts and are of the form [rr:column v], [rr:constant c] or [rr:template s].

Given a triple map m with a logical table (SQL query) R, we construct a selection $\sigma_{\neg \mathit{isNull}(v_1)} \cdots \sigma_{\neg \mathit{isNull}(v_k)} R$, where $v_1, \ldots, v_k$ are the *referenced columns* of m (attributes of R in the term maps in m)—this is done to exclude tuples that contain *null* [10]. To construct tr_m, the selection filter is prefixed with projection $\pi_{\mathit{subj},\mathit{pred},\mathit{obj}}$

and, for each of the three term maps, either with renaming (e.g., with $\rho_{obj/v}$ if the object map is of the form [rr:column v]) or with value creation (if the term map is of the form [rr:constant c] or [rr:template s]; in the latter case, we use the built-in string concatenation function ||). For instance, the mapping _:m1 from Example 10 is converted to the SQL query

```
SELECT ('/GradStudent' || id) AS subj, 'ub:UGDegreeFrom' AS pred,
          ('/Uni' || degreeuniid) AS obj FROM students
WHERE (id IS NOT NULL) AND (degreeuniid IS NOT NULL) AND (stype=1).
```

Given an R2RML mapping $\mathcal{M}$, we set $\mathsf{tr}_{\mathcal{M}}(\mathit{triple}) = \bigcup_{m \in \mathcal{M}} \mathsf{tr}_m$.

Proposition 11. *For any R2RML mapping $\mathcal{M}$ and data instance D, $t \in \|\mathsf{tr}_{\mathcal{M}}(\mathit{triple})\|_D$ if and only if $t \in \mathit{triple}(G_{D,\mathcal{M}})$.*

Finally, given a graph pattern P and an R2RML mapping $\mathcal{M}$, we define $\mathsf{tr}_{\mathcal{M}}(\boldsymbol{\tau}(P))$ to be the result of replacing every occurrence of the relation *triple* in the query $\boldsymbol{\tau}(P)$, constructed in Section 3, with $\mathsf{tr}_{\mathcal{M}}(\mathit{triple})$. By Theorem 7 and Proposition 11, we obtain:

Theorem 12. *For any graph pattern P, R2RML mapping $\mathcal{M}$ and data instance D, $\|P\|_{G_{D,\mathcal{M}}} = \|\mathsf{tr}_{\mathcal{M}}(\boldsymbol{\tau}(P))\|_D$.*

3.3 Optimising SQL Translation

The straightforward application of $\mathsf{tr}_{\mathcal{M}}$ to $\boldsymbol{\tau}(P)$ can result in a very complex SQL query. We now show that such queries can be optimised by the following techniques:

- choosing matching tr_m from $\mathsf{tr}_{\mathcal{M}}(\mathit{triple})$, for each occurrence of *triple* in $\boldsymbol{\tau}(P)$;
- using the distributivity of $\bowtie$ over $\cup$ and removing sub-queries with *incompatible IRI templates* and *de-IRIing* join conditions;
- functional dependencies (e.g., primary keys) for self-join elimination [6,18,29,30].

To illustrate, suppose we are given a mapping $\mathcal{M}$ containing _:m1 from Example 10 and the following triple maps (which are a simplified version of those in Section 5):

```
_:m2 a rr:TripleMap;
      rr:logicalTable [ rr:sqlQuery "SELECT * FROM students WHERE stype=0" ];
      rr:subjectMap [ rr:template "/UGStudent{id}"; rr:class ub:Student ].
_:m3 a rr:TripleMap;
      rr:logicalTable [ rr:sqlQuery "SELECT * FROM students WHERE stype=1" ];
      rr:subjectMap [ rr:template "/GradStudent{id}"; rr:class ub:Student ].
```

which generate undergraduate and graduate students (both are instances of ub:Student, but their IRIs are constructed using different templates [16]). Consider the following query (a fragment of q_2^{obg} from Section 5):

SELECT $?x$ $?y$ WHERE { $?x$ rdf:type ub:Student. $?x$ ub:UGDegreeFrom $?y$ }.

The translation $\boldsymbol{\tau}$ of its BGP (after the SPARQL JOIN optimisation of Section 3.1) is

$$(\pi_x \rho_{x/\mathit{subj}} \sigma_{(\mathit{pred}=\text{rdf:type}) \wedge (\mathit{obj}=\text{ub:Student})}\, \mathit{triple}) \bowtie \\ (\pi_{x,y} \rho_{x/\mathit{subj}} \rho_{y/\mathit{obj}} \sigma_{\mathit{pred}=\text{ub:UGDegreeFrom}}\, \mathit{triple})$$

First, since *triple* always occurs in the scope of some selection operation σ_F, we can choose only those elements in $\bigcup_{m \in \mathcal{M}} \mathsf{tr}_m$ that have matching values of *pred* and/or *obj*. In our example, the first occurrence of *triple* is replaced by $\mathsf{tr}_{\_:\mathsf{m2}} \cup \mathsf{tr}_{\_:\mathsf{m3}}$, and the second one by $\mathsf{tr}_{\_:\mathsf{m1}}$. This results in the natural join of the following union, denoted A:

```
(SELECT DISTINCT '/UGStudent' || id AS x FROM students
 WHERE (id IS NOT NULL) AND (stype=0))
UNION (SELECT DISTINCT '/GradStudent' || id AS x FROM students
         WHERE (id IS NOT NULL) AND (stype=1))
```

and of the following query, denoted B:

```
SELECT DISTINCT '/GradStudent' || id AS x, '/Uni' || degreeuniid AS y FROM students
WHERE (id IS NOT NULL) AND (degreeuniid IS NOT NULL) AND (stype=1)
```

Second, observe that the IRI template in B is compatible only with the second component of A. Moreover, since the two compatible templates coincide, we can *de-IRI* the join, namely, replace the join over the constructed strings (A.x = B.x) by the join over the numerical attributes (A.id = B.id), which results in a more efficient query:

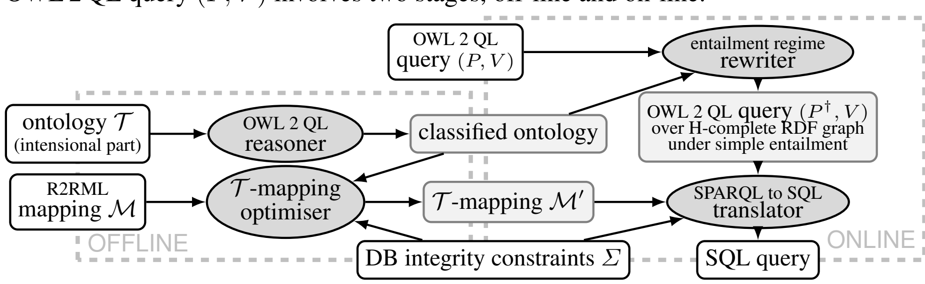
```
SELECT DISTINCT A.x, B.y FROM
   (SELECT id, '/GradStudent' || id AS x FROM students
    WHERE (id IS NOT NULL) AND (stype=1)) A
   JOIN
      (SELECT id, '/GradStudent' || id AS x, '/Uni' || degreeuniid AS y FROM students
       WHERE (id IS NOT NULL) AND (degreeuniid IS NOT NULL) AND (stype=1)) B
   ON A.id = B.id
```

Finally, by using self-join elimination and the fact that id and stype are the composite primary key in students, we obtain the query (without DISTINCT as x is unique)

```
SELECT '/GradStudent' || id AS x, '/Uni' || degreeuniid AS y FROM students
WHERE (degreeuniid IS NOT NULL) AND (stype=1)
```

4 Putting it all Together

The techniques introduced above suggest the following architecture to support answering SPARQL queries under the OWL 2 QL entailment regime with data instances stored in a database. Suppose we are given an ontology with an intensional part $\mathcal{T}$ and an extensional part stored in a database, D, over a schema Σ. Suppose also that the languages of Σ and $\mathcal{T}$ are connected by an R2RML mapping $\mathcal{M}$. The process of answering a given OWL 2 QL query (P, V) involves two stages, off-line and on-line.



The *off-line* stage takes $\mathcal{T}$, $\mathcal{M}$ and Σ and proceeds via the following steps:
❶ An OWL 2 QL reasoner is used to obtain a complete class / property hierarchy in $\mathcal{T}$.

❷ The composition $\mathcal{M}^{\mathcal{T}}$ of $\mathcal{M}$ with the class and property hierarchy in $\mathcal{T}$ is taken as an initial $\mathcal{T}$-mapping. Recall [29] that a mapping $\mathcal{M}'$ is a *$\mathcal{T}$-mapping over Σ* if, for any data instance D satisfying Σ, the *virtual* (not materialised) RDF graph $G_{D,\mathcal{M}'}$ obtained by applying $\mathcal{M}'$ to D contains all class and property assertions α with $(\mathcal{T}, G_{D,\mathcal{M}'}) \models \alpha$. As a result, $G_{D,\mathcal{M}'}$ is complete with respect to the class and property hierarchy in $\mathcal{T}$ (or H-complete), which allows us to avoid reasoning about class and property inclusions (in particular, inferences that involve property domains and ranges) at the query rewriting step ❹ and drastically simplify rewritings (see [29] for details).

❸ The initial $\mathcal{T}$-mapping $\mathcal{M}^{\mathcal{T}}$ is then optimised by (*i*) eliminating redundant triple maps detected by query containment with inclusion dependencies in Σ, (*ii*) eliminating redundant joins in logical tables using the functional dependencies in Σ, and (*iii*) merging sets of triple maps by means of interval expressions or disjunctions in logical tables (see [29] for details). Let $\mathcal{M}'$ be the resulting $\mathcal{T}$-mapping over Σ.

The *on-line* stage takes an OWL 2 QL query (P, V) as an input and proceeds as follows:

❹ The graph pattern P and $\mathcal{T}$ are rewritten to the OWL 2 QL graph pattern $P^\dagger$ over the H-complete virtual RDF graph $G_{D,\mathcal{M}'}$ *under simple entailment* by applying the classified ontology of step ❶ to instantiate class and property variables and then using a query rewriting algorithm (e.g., the tree-witness rewriter of [29]); see Theorem 4.

❺ The graph pattern $P^\dagger$ is transformed to the SQL query $\boldsymbol{\tau}(P^\dagger)$ over the 3-column representation *triple* of the RDF graph (Theorem 7). Next, the query $\boldsymbol{\tau}(P^\dagger)$ is unfolded into the SQL query $\mathsf{tr}_{\mathcal{M}'}(\boldsymbol{\tau}(P^\dagger))$ over the original database D (Theorem 12). The unfolded query is optimised using the techniques similar to the ones employed in step ❸.

❻ The optimised query is executed by the database.

As follows from Theorems 4, 7 and 12, the resulting query gives us all correct answers to the original OWL 2 QL query (P, V) over $\mathcal{T}$ and D with the R2RML mapping $\mathcal{M}$.

5 Evaluation

The architecture described above has been implemented in the open-source OBDA system *Ontop*[4]. We evaluated its performance using the OWL 2 QL version of the Lehigh University Benchmark LUBM [16]. The ontology contains 43 classes, 32 object and data properties and 243 axioms. The benchmark also includes a data generator and a set of 14 queries q_1–q_{14}. We added 7 queries with second-order variables ranging over class and property names: q_4', q_4'', q_9', q_9'' derived from q_4 and q_9, and $q_2^{obg}, q_4^{obg}, q_{10}^{obg}$ taken from [19]. The LUBM data generator produces an OWL file with class and property assertions. To store the assertions in a database, we created a database schema with 11 relations and an R2RML mapping with 89 predicate-object maps. For instance, the information about undergraduate and graduate students (id, name, etc.) from Example 10 is collected in the relation students, where the attribute stype distinguishes between the types of students (stype is known as a discriminant column in databases); more details including primary and foreign keys and indexes are provided in the full version.

We experimented with the data instances LUBM$_n$, $n = 1, 9, 20, 50, 100, 200, 500$ (where n specifies the number of universities; LUBM$_1$ and LUBM$_9$ were used in [19]).

[4] `http://ontop.inf.unibz.it`

Q	LUBM$_1$				LUBM$_9$			LUBM$_{100}$		LUBM$_{200}$	LUBM$_{500}$
	O	OB$_H$	OB$_P$	P	O	OB$_H$	P	O	P	O	O
q_1	2	8	29	1	3	97	1	3	1	3	2
q_2	2	25	11 137	19	3	2 531	256	16	30 593	36	88
q_3	1	6	86	9	2	78	158	2	2 087	63	12
q_4	13	7	19	14	15	44	164	27	2 093	24	22
q_5	16	12	4 451	10	22	98	158	32	2 182	28	23
q_6	455	27	32	21	5 076	411	317	58 968	10 781	123 578	434 349
q_7	5	21	34 005	10	6	429	157	8	2 171	8	9
q_8	726	195	95 875	80	760	917	192	796	2 131	820	855
q_9	60	972	168 978	78	668	189 126	857	7 466	12 125	15 227	44 598
q_{10}	2	6	126	9	3	97	158	2	2 134	3	2
q_{11}	4	5	58	10	6	43	160	11	2 093	18	44
q_{12}	3	4	19	15	4	70	236	3	2 114	5	5
q_{13}	6	4	67	8	7	40	157	14	2 657	38	58
q_{14}	91	20	24	15	1 168	329	287	13 524	4 457	29 512	92 376
q_4'	93	58	190	46	99	98	767	92	4 422	95	107
q_4''	108	21	35	63	122	72	719	115	9 179	108	127
q_9'	257	716	91 855	174	4 686	40 575	1 385	54 092	19 945	115 110	295 228
q_9''	557	951	65 916	102	6 093	178 401	1 214	67 123	19 705	151 376	356 176
q_2^{obg}	150	30	57 141	29	9 992	520	348	39 477	5 411	79 351	206 061
q_4^{obg}	6	7	241	25	31	40	273	7	3 969	7	494
q_{10}^{obg}	641	760	31 269	253	6 998	149 191	2 258	163 308	17 929	174 362	459 669
start up	3.1s	13.6s	7.7s	3.6s	3.1s	80m33s	18s	3.1s	3m23s	3.1s	3.1s
data load	10s	n/a	n/a	n/a	15s	n/a	n/a	1m56s	n/a	3m35s	10m17s

Table 1. Start up time, data loading time (in s) and query execution time (in ms): O is *Ontop*, OB$_H$ and OB$_P$ are OWL-BGP with Hermit and Pellet, respectively, and P is standalone Pellet.

Here we only show the results for $n = 1, 9, 100, 200, 500$ containing 103k, 1.2M, 14M, 28M and 69M triples, respectively; the complete table can be found in the full version. All the materials required for the experiments are available online[5]. We compared *Ontop* with two other systems, OWL-BGP r123 [19] and Pellet 2.3.1 [31] (Stardog and OWLIM are incomplete for the OWL 2 QL entailment regime). OWL-BGP requires an OWL 2 reasoner as a backend; as in [19], we employed HermiT 1.3.8 [14] and Pellet 2.3.1. The hardware was an HP Proliant Linux server with 144 cores @3.47GHz, 106GB of RAM and a 1TB 15k RPM HD. Each system used a single core and was given 20 GB of Java 7 heap memory. *Ontop* used MySQL 5.6 database engine.

The evaluation results are given in Table 1. OWL-BGP and Pellet used significantly more time to start up (last but one row) because they do not rely on query rewriting and require costly pre-computations. OWL-BGP failed to start on LUBM$_9$ with Pellet and on LUBM$_{20}$ with HermiT; Pellet ran out of memory after 10hrs loading LUBM$_{200}$. For *Ontop*, the start up is the off-line stage described in Section 4; it does not include the time of loading the data into MySQL, which is specified in the last row of Table 1 (note that the data is loaded only once, not every time *Ontop* starts; moreover, this could be improved with CSV loading and delayed indexing rather than SQL dumps we used).

On queries q_1–q_{14}, *Ontop* generally outperforms OWL-BGP and Pellet. Due to the optimisations, the SQL queries generated by *Ontop* are very simple, and MySQL is able to execute them efficiently. This is also the case for large datasets, where *Ontop* is able to maintain almost constant times for many of the queries. Notable exceptions are q_6, q_8 and q_{14} that return a very large number (hundreds of thousands) of results (low

[5] `https://github.com/ontop/iswc2014-benchmark`

selectivity). A closer inspection reveals that execution time is mostly spent on fetching the results from disk. On the queries with second-order variables, the picture is mixed. While indeed these queries are not the strongest point of *Ontop* at the moment, we see that in general the performance is good. Although Pellet outperforms *Ontop* on small datasets, only *Ontop* is able to provide answers for very large datasets. For second-order queries with high selectivity (e.g., q_4' and q_4'') and large datasets, the performance of *Ontop* is very good while the other systems fail to return answers.

6 Conclusions

In this paper, we gave both a theoretical background and a practical implementation of a procedure for answering SPARQL 1.1 queries under the OWL 2 QL direct semantics entailment regime in the scenario where data instances are stored in a relational database whose schema is connected to the language of the given OWL 2 QL ontology via an R2RML mapping. Our main contributions can be summarised as follows:

– We defined an entailment regime for SPARQL 1.1 corresponding to the OWL 2 QL profile of OWL 2 (which was specifically designed for ontology-based data access).
– We proved that answering SPARQL queries under this regime is reducible to answering SPARQL queries under simple entailment (where no reasoning is involved).
– We showed how to transform such SPARQL queries to equivalent SQL queries over an RDF representation of the data, and then unfold them, using R2RML mappings, into SQL queries over the original relational data.
– We developed optimisation techniques to substantially reduce the size and improve the quality of the resulting SQL queries.
– We implemented these rewriting and optimisation techniques in the OBDA system *Ontop*. Our initial experiments showed that *Ontop* generally outperforms reasoner-based systems, especially on large data instances.

Some aspects of SPARQL 1.1 (such as RDF types, property paths, aggregates) were not discussed here and are left for future work.

Acknowledgements. Our work was supported by EU project Optique. We thank S. Komla-Ebri for help with the experiments, and I. Kollia and B. Glimm for discussions.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Angles, R., Gutierrez, C.: The expressive power of SPARQL. In: Proc. of ISWC. LNCS, vol. 5318, pp. 114–129. Springer (2008)
3. Bornea, M., Dolby, J., Kementsietsidis, A., Srinivas, K., Dantressangle, P., Udrea, O., Bhattacharjee, B.: Building an efficient RDF store over a relational database. In: Proc. of SIGMOD 2013, pp. 121–132. ACM (2013)
4. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D.F.: The MASTRO system for ontology-based data access. Semantic Web 2(1), 43–53 (2011)
5. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. J. of Automated Reasoning 39(3), 385–429 (2007)

6. Chakravarthy, U.S., Grant, J., Minker, J.: Logic-based approach to semantic query optimization. ACM Transactions on Database Systems 15(2), 162–207 (1990)
7. Chebotko, A., Lu, S., Fotouhi, F.: Semantics preserving SPARQL-to-SQL translation. Data Knowl. Eng. 68(10), 973–1000 (2009)
8. Chortaras, A., Trivela, D., Stamou, G.: Optimized query rewriting for OWL 2 QL. In: Proc. of CADE-23. LNCS, vol. 6803, pp. 192–206. Springer (2011)
9. Cyganiak, R.: A relational algebra for SPARQL. Tech. Rep. HPL-2005-170, HP Labs (2005)
10. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF Mapping Language (September 2012), `http://www.w3.org/TR/r2rml`
11. Dolby, J., Fokoue, A., Kalyanpur, A., Ma, L., Schonberg, E., Srinivas, K., Sun, X.: Scalable grounded conjunctive query evaluation over large and expressive knowledge bases. In: Proc. of ISWC. LNCS, vol. 5318, pp. 403–418. Springer (2008)
12. Eiter, T., Ortiz, M., Šimkus, M., Tran, T.K., Xiao, G.: Query rewriting for Horn-SHIQ plus rules. In: Proc. of AAAI. AAAI Press (2012)
13. Elliott, B., Cheng, E., Thomas-Ogbuji, C., Özsoyoglu, Z.M.: A complete translation from SPARQL into efficient SQL. In: Proc. of IDEAS. pp. 31–42. ACM (2009)
14. Glimm, B., Horrocks, I., Motik, B., Stoilos, G.: Optimising ontology classification. In: Proc. of ISWC, part I. LNCS, vol. 6496, pp. 225–240. Springer (2010)
15. Gottlob, G., Orsi, G., Pieris, A.: Ontological queries: Rewriting and optimization. In: Proc. of ICDE. pp. 2–13. IEEE Computer Society (2011)
16. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. J. of Web Semantics 3(2–3), 158–182 (2005)
17. Heymans, S. *et al.*: Ontology reasoning with large data repositories. In: Ontology Management, Semantic Web, Semantic Web Services, and Business Applications. Springer (2008)
18. King, J.J.: Query Optimization by Semantic Reasoning. Ph.D. thesis, Stanford, USA (1981)
19. Kollia, I., Glimm, B.: Optimizing SPARQL query answering over OWL ontologies. J. of Artificial Intelligence Research 48, 253–303 (2013)
20. König, M., Leclère, M., Mugnier, M.-L., Thomazo, M.: On the exploration of the query rewriting space with existential rules. In: Proc. of RR. LNCS. pp. 123–137. Springer (2013)
21. Kontchakov, R., Rodríguez-Muro, M., Zakharyaschev, M.: Ontology-based data access with databases: A short course. In: Reasoning Web. LNCS, vol. 8067, pp. 194–229. Springer (2013)
22. Lutz, C., Seylan, I., Toman, D., Wolter, F.: The combined approach to OBDA: Taming role hierarchies using filters. In: Proc. of ISWC. LNCS, vol. 8218, pp. 314–330. Springer (2013)
23. Pérez-Urbina, H., Rodríguez-Díaz, E., Grove, M., Konstantinidis, G., Sirin, E.: Evaluation of query rewriting approaches for OWL 2. In: SSWS+HPCSW. CEUR-WS, vol. 943 (2012)
24. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. on Data Semantics X, 133–173 (2008)
25. Polleres, A.: From SPARQL to rules (and back). In: Proc. WWW. pp. 787–796. ACM (2007)
26. Polleres, A., Wallner, J.P.: On the relation between SPARQL 1.1 and Answer Set Programming. J. of Applied Non-Classical Logics 23(1–2), 159–212 (2013)
27. Priyatna, F., Corcho, O., Sequeda, J.: Formalisation and experiences of R2RML-based SPARQL to SQL query translation using Morph. In: Proc. of WWW. pp. 479–490 (2014)
28. Rodríguez-Muro, M., Hardi, J., Calvanese, D.: Quest: Efficient SPARQL-to-SQL for RDF and OWL. In: Proc. of the ISWC 2012 P&D Track. vol. 914. CEUR-WS.org (2012)
29. Rodríguez-Muro, M., Kontchakov, R., Zakharyaschev, M.: Ontology-based data access: Ontop of databases. In: Proc. of ISWC. LNCS, vol. 8218, pp. 558–573. Springer (2013)
30. Sequeda, J.F., Miranker, D.P.: Ultrawrap: SPARQL execution on relational data. J. of Web Semantics 22, 19–39 (2013)
31. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL Reasoner. J. of Web Semantics 5(2), 51–53 (2007)
32. Zemke, F.: Converting SPARQL to SQL. Tech. rep., Oracle Corp. (2006)