Rules and Ontology Based Data Access

Guohui Xiao[1], Martin Rezk[1], Mariano Rodríguez-Muro[2], and Diego Calvanese[1]

[1]Faculty of Computer Science, Free University of Bozen-Bolzano, Italy
[2]IBM Watson Research Center, USA

Abstract. In OBDA an ontology defines a high level global vocabulary for user queries, and such vocabulary is mapped to (typically relational) databases. Extending this paradigm with rules, e.g., expressed in SWRL or RIF, boosts the expressivity of the model and the reasoning ability to take into account features such as recursion and n-ary predicates. We consider evaluation of SPARQL queries under rules with linear recursion, which in principle is carried out by a 2-phase translation to SQL: (1) The SPARQL query, together with the RIF/SWRL rules, and the mappings is translated to a Datalog program, possibly with linear recursion; (2) The Datalog program is converted to SQL by using recursive common table expressions. Since a naive implementation of this translation generates inefficient SQL code, we propose several optimisations to make the approach scalable. We implement and evaluate the techniques presented here in the *Ontop* system. To the best of our knowledge, this results in the first system supporting all of the following W3C standards: the OWL 2 QL ontology language, R2RML mappings, SWRL rules with linear recursion, and SPARQL queries. The preliminary but encouraging experimental results on the NPD benchmark show that our approach is scalable, provided optimisations are applied.

1 Introduction

In Ontology Based Data Access (OBDA) [5], the objective is to access data trough a conceptual layer. Usually, this conceptual layer is expressed in the form of an OWL or RDFS ontology, and the data is stored in relational databases. The terms in the conceptual layer are mapped to the data layer using so-called globas-as-view (GAV) mappings, associating to each element of the conceptual layer a (possibly complex) query over the data sources. GAV mappings have been described as Datalog rules in the literature [17] and formalized in the R2RML W3C standard [8]. Independently of the mapping language, these rules entail a *virtual* RDF graph that uses the ontology vocabulary. This virtual graph can then be queried using an RDF query language such as SPARQL.

There are several approaches for query answering in the context of OBDA, and a number of techniques have been proposed [17,16,13,21,9,3]. One of such techniques, and the focus of this paper, is query answering by *query rewriting*. That is, answer the queries posed by the user (e.g., SPARQL queries) by translating them into queries over the database (e.g., SQL). This kind of technique has several desirable features; notably, since all data remains in the original source there is no redundancy, the system immediately reflects any changes in the data, well-known optimizations for relational databases can be used, etc. It has been shown that through this technique one can obtain performance comparable or sometimes superior to other approaches when the ontology language is restricted to OWL 2 QL [22]. While the OWL 2 QL specification (which subsumes RDFS in expressive power) offers a good balance between expressivity and performance, there are many scenarios where this expressive power is not enough.

As a motivating example and to illustrate the main concepts in this paper, suppose we have a (virtual) RDF graph over a database with information about direct flights between locations and their respective cost. Suppose we have a `flight` relation in the database, and we want to find out all the possible (direct and non-direct) routes between two locations such that the total cost is less than 100 Euros. This problem is a particular instance of the well-known reachability problem, where we need to be able to compute the transitive closure over the `flight` relation respecting the constraint on the flight cost. While SPARQL 1.1 provides *path expressions* that can be used to express the transitive closure of a property, it may be cumbersome and prone to errors, especially in the presence of path constraints such as the cost in our example.

Computational complexity results show that unless we limit the form of the allowed rules, on-the-fly query answering by rewriting into SQL Select-Project-Join (SPJ) queries is not possible [6,2]. However, as target language for query rewriting, typically only a fragment of the expressive power of SQL99 has been considered, namely unions of SPJ SQL queries. We propose here to go beyond this expressive power, and we advocate the use of SQL99's Common Table Expressions (CTEs) to obtain a form of linear recursion in the rewriting target language. In this way, we can deal with recursive rules at the level of the ontology, and can reuse existing query rewriting optimisations developed for OBDA to provide efficient query rewriting into SQL99. The languages that we target are those that are used more extensively in the context of OBDA for Semantic Web application, i.e., RIF and SWRL as rule language, SPARQL 1.0 as query language, and R2RML as relational databases to RDF mapping language.

The contributions of this paper can be summarized as follows: *(i)* We provide translations from SWRL, R2RML, and SPARQL into relational algebra extended with a fixed-point operator that can be expressed in SQL99's Common Table Expressions (CTEs); *(ii)* We show how to extend existing OBDA optimisation techniques that have been proven effective in the OWL 2 QL setting to this new context. In particular, we show that so called *T-mappings* for recursive programs exist and how to construct them. *(iii)* We provide an implementation of such technique in the open source OBDA system *Ontop*, making it the first system of its kind to support all the following W3C recommendations: OWL 2 QL, R2RML, SPARQL, and SWRL; *(iv)* We provide a preliminary evaluation of the techniques using an extension of the NPD benchmark (a recently developed OWL 2 QL benchmark) with rules, and show that the proposed solution competes and sometimes outperforms existing triple stores.

2 Preliminaries

2.1 RDF

The Resource Description Framework (RDF) is a standard model for data interchange on the Web [15]. The language of RDF contains the following pairwise disjoint and countably infinite sets of symbols: **I** for *IRIs*, **L** for *RDF literals*, and **B** for *blank nodes*. *RDF terms* are elements of the set $\mathbf{T} = \mathbf{I} \cup \mathbf{B} \cup \mathbf{L}$. An *RDF knowledge base* (also called *RDF graph*) is a collection of triples of the form (s, p, o), where $s \in \mathbf{I}$, $p \in \mathbf{I} \cup \mathbf{B}$, and $o \in \mathbf{T}$. A triple (s, p, o) intuitively expresses that s and o are related by p; when p is the special role `rdf:type`, the triple $(s, \texttt{rdf:type}, o)$ means that s is an instance of o.

It is sometimes convenient to define conversions between RDF graphs and sets of (Datalog) facts. Thus, given an RDF graph G, the corresponding set of Datalog facts is:

$$\mathcal{A}(G) = \{o(s) \mid (s, \texttt{rdf:type}, o) \in G\} \cup \{p(s, o) \mid (s, p, o) \in G, p \neq \texttt{rdf:type}\}$$

And given a set A of facts, the corresponding RDF graph is:

$$\mathcal{G}(A) = \{(s, \texttt{rdf:type}, o) \mid o(s) \in A\} \cup \{(s, p, o) \mid p(s, o) \in A\}$$

Note that $\mathcal{G}(A)$ discards the facts that are not unary or binary.

2.2 SPARQL

SPARQL is the standard RDF query language. For formal purposes we will use the algebraic syntax of SPARQL similar to the one in [18] and defined in the standard[1]. However, to ease the understanding, we will often use graph patterns (the usual SPARQL syntax) in the examples. The SPARQL language that we consider shares with RDF the set of symbols: constants, blank nodes, IRIs, and literals. In addition, it adds a countably infinite set **V** of variables. The *SPARQL algebra* is constituted by the following graph pattern operators (written using prefix notation): BGP (basic graph pattern), $Join$, $LeftJoin$, $Filter$, and $Union$. A *basic graph pattern* is a statement of the form:$BGP(s, p, o)$. In the standard, a BGP can contain several triples, but since we include here the join operator, it suffices to view BGPs as the result of $Join$ of its constituent triple patterns. Observe that the only difference between blank nodes and variables in BGPs, is that the former do not occur in solutions. So, to ease the presentation, we assume that BGPs contain no blank nodes. Algebra operators can be nested freely. Each of these operators returns the result of the sub-query it describes.

Definition 1 (SPARQL Query). *A* SPARQL query *is a pair (V, P), where V is a set of variables, and P is a SPARQL algebra expression in which all variables of V occur.*

We will often omit V when it is understood from the context . A *substitution*, θ, is a *partial* function $\theta \colon \mathbf{V} \mapsto \mathbf{T}$. The domain of θ, denoted by $dom(\theta)$, is the subset of **V** where θ is defined. Here we write substitutions using postfix notation. When a query (V, P) is evaluated, the result is a set of substitutions whose domain is contained in V. For space reasons, we omit the semantics of SPARQL, and refer to [11] for the specification of how to compute the answer of a query Q over an RDF graph G, which we denote as $[\![Q]\!]_G$.

Example 1 (Flights, continued). Consider the flight example in the introduction. The low cost flights from Bolzano can be retrieved by the query:

[1] `http://www.w3.org/TR/rdf-sparql-query/#sparqlAlgebra`

```
Select ?x  Where {
  ?x :tripPlanFrom :Bolzano .  ?x :tripPlanTo ?y .
  ?x :tripPlanPrice ?z .  Filter(?z < 100)
}
```

The corresponding SPARQL algebra expression is as follows:

```
Filter (?z < 100)(
 Join(BGP(?x :tripPlanFrom :Bolzano .)
  Join(BGP(?x :tripPlanTo ?y .) BGP(?x :tripPlanPrice ?z .))))
```

2.3 Rules: RIF and SWRL

We describe now two important rule languages, SWRL and RIF, and the semantics of their combination with RDF graphs.

The Semantic Web Rule Language (SWRL) is a widely used Semantic Web language combining a DL ontology component with rules[2]. Notice that the SWRL language allows only for the use of unary and binary predicates. SWRL is implemented in many systems, such as, Pellet, Stardog, and HermiT.

The Rule Interchange Format (RIF) is a W3C recommendation [12] defining a language for expressing rules. The standard RIF dialects are Core, BLD, and PRD. RIF-Core provides "safe" positive Datalog with built-ins; RIF-BLD (Basic Logic Dialect) is positive Horn logic, with equality and built-ins; RIF-PRD (Production Rules Dialect) adds a notion of forward-chaining rules, where a rule fires and then performs some action. In this paper we focus on RIF-Core [4], which is equivalent to Datalog without negation, but supports an F-Logic style frame-like syntax: $s[p_1 \to o_1, p_2 \to o_2, \dots, p_n \to o_n]$ is a shorthand for the conjunction of atoms $\bigwedge_{p_i = \texttt{rdf:type}} o_i(s) \wedge \bigwedge_{p_i \neq \texttt{rdf:type}} p_i(s, o_i)$. Observe that the RIF language allows for additional n-ary Datalog predicates besides the unary concept names and binary role names from the RDF vocabulary. In this paper, we make the restriction that variables cannot be used in the place of the predicates. For instance, neither $s[?p \to o]$ nor $s[\texttt{rdf:type} \to ?o]$ are allowed.

For the sake of simplicity, in the following we will use Datalog notation, where (SWRL or RIF) rules are simply written as

$$l_0 \text{ :- } l_1, \dots, l_m$$

where each l_i is an atom. Therefore we refer to a set of rules as Datalog rules. Recall that a Datalog program Π that does not contain negation has a unique minimal model, which can be computed via a repeated exhaustive application of the rules in it in a bottom-up fashion [14]. We denote such model, $\mathsf{MM}(\Pi)$.

An *RDF-rule combination* is a pair (G, Π), where G is an RDF graph and Π is a set of Datalog rules.

Definition 2. *The* RDF graph induced by *an RDF-rule combination* (G, Π) *is defined as* $\mathcal{G}(\mathsf{MM}(\mathcal{A}(G) \cup \Pi))$.

[2] `http://www.w3.org/Submission/SWRL/`

Example 2 (Example 1, continued). The following rules model the predicates `plan`, representing the transitive closure of flights, including the total price of the trip, and `tripPlanFrom/To/Price`, which project `plan` into triples:

$$\begin{array}{lll}
\texttt{plan}(\mathit{from}, \mathit{to}, \mathit{price}, \mathit{plan\_url}) & \text{:-} & \texttt{flightFrom}(\mathit{fid}, \mathit{from}),\ \texttt{flightTo}(\mathit{fid}, \mathit{to}), \\
 & & \texttt{flightPrice}(\mathit{fid}, \mathit{price}), \\
 & & \mathit{plan\_url} = \texttt{CONCAT}(\texttt{"http://flight/"}, \mathit{fid}) \\
\texttt{plan}(\mathit{from}, \mathit{to}, \mathit{price}, \mathit{plan\_url}) & \text{:-} & \texttt{plan}(\mathit{from}, \mathit{to_1}, \mathit{price_1}, \mathit{plan\_url_1}), \\
 & & \texttt{flightFrom}(\mathit{fid}, \mathit{to_1}),\ \texttt{flightTo}(\mathit{fid}, \mathit{to}), \\
 & & \texttt{flightPrice}(\mathit{fid}, \mathit{price_2}), \\
 & & \mathit{price} = \mathit{price_1} + \mathit{price_2}, \\
 & & \mathit{plan\_url} = \texttt{CONCACT}(\mathit{plan\_url_1}, \texttt{"/"}, \mathit{fid}) \\
\texttt{tripPlanFrom}(\mathit{plan\_url}, \mathit{from}) & \text{:-} & \texttt{plan}(\mathit{from}, \mathit{to}, \mathit{price}, \mathit{plan\_url}) \\
\texttt{tripPlanTo}(\mathit{plan\_url}, \mathit{to}) & \text{:-} & \texttt{plan}(\mathit{from}, \mathit{to}, \mathit{price}, \mathit{plan\_url}) \\
\texttt{tripPlanPrice}(\mathit{plan\_url}, \mathit{price}) & \text{:-} & \texttt{plan}(\mathit{from}, \mathit{to}, \mathit{price}, \mathit{plan\_url})
\end{array}$$

Observe that rules not only boost the modelling capabilities by adding recursion, but also allow for a more elegant and succinct representation of the domain using n-ary predicates.

2.4 SPARQL and Rules: Entailment Regime

The RIF entailment regime specifies how RIF entailment can be used to redefine the evaluation of basic graph patterns. The evaluation of complex clauses is computed by combining already computed solutions in the usual way. Therefore, in this section we can restrict the attention to queries that consist of a single BGP.

The semantics provided in [10] is defined in terms of pairs of RIF and RDF interpretations. These models are then used to define satisfiability and entailment in the usual way. Combined entailment extends both entailment in RIF and entailment in RDF. To ease the presentation, we will present a simplified version of such semantics based on Datalog models seen as RDF graphs (c.f. Section 2.1).

Definition 3. *Let Q be a BGP, G an RDF graph, and Π a set of rules. The evaluation of Q over G and Π, denoted $[\![Q]\!]_{G,\Pi}$, is defined as the evaluation of Q over the induced RDF graph of (G, Π), that is*

$$[\![Q]\!]_{G,\Pi} = [\![Q]\!]_{\mathcal{G}(\mathit{MM}(\mathcal{A}(G) \cup \Pi))}.$$

Example 3 (Example 2, continued). Suppose we have the following triples in our RDF graph:

```
AF22 :flightFrom :Bolzano .    AF23 :flightTo :Dublin .
AF22 :flightTo   :Milano .     AF22 :flightPrice 45 .
AF23 :flightFrom :Milano .     AF23 :flightPrice 45 .
```

It is easy to see that these triples, together with the rules in Example 2 "extend" the original RDF with the following triples:

```
http://flight/AF22/AF23 :tripPlanFrom :Bolzano .
http://flight/AF22/AF23 :tripPlanTo   :Dublin .
http://flight/AF22/AF23 :tripPlanPrice 90 .
```

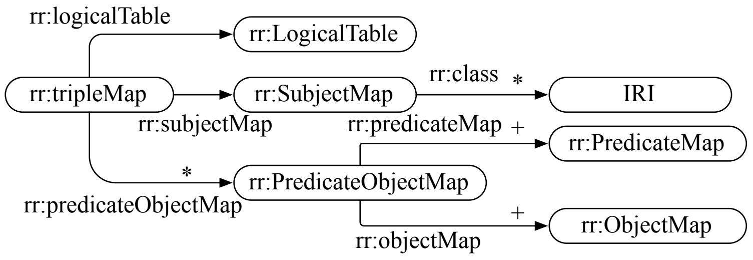

Fig. 1. A well formed R2RML mapping node

This implies that we get the following two substitutions by evaluating the query in Example 1: $\{x \mapsto \texttt{http://flight/AF22}\}, \{x \mapsto \texttt{http://flight/AF22/AF23}\}$.

2.5 R2RML: Mapping Databases to RDF

R2RML is a W3C standard [8] defining a language for mapping relational databases into RDF data. Such mappings expose the relational data as RDF triples, using a structure and vocabulary chosen by the mapping author.

An R2RML mapping is expressed as an RDF graph (in Turtle syntax), where a well-formed mapping consists of one or more trees called *triple maps* with a structure as shown in Figure 1. Each tree has a root node, called *triple map node*, which is linked to exactly one *logical table* node, one *subject map* node and one or more *predicate object map* nodes. Intuitively, each triple map states how to construct a set of triples (subject, predicate, object) using the information contained in the logical table (specified as an SQL query).

The R2RML syntax is rather verbose, therefore, due to the space limitations, in this paper we represent triple maps using standard Datalog rules of the form:

$$\mathit{predicate}(\mathit{subject}, \mathit{object}) \text{ :- } \mathit{body} \qquad\qquad \mathit{concept}(\mathit{subject}) \text{ :- } \mathit{body}$$

where body is a conjunction of atoms that refers to the database relations, possibly making use of auxiliary relations representing the SQL query of the mapping, when the semantics of such query cannot be captured in Datalog. For the formal translation from R2RML to Datalog, we refer to [23].

Example 4 (Example 2, continued). We present the R2RML rules mapping a relational database to the relations `flightFrom` and `flightPrice`. Recall that the relations `tripPlanFrom`, `tripPlanTo`, etc. are defined by rules. Suppose we have a table `flight` in the database, with attributes: id, $departure$, $arrival$, $segment$, and $cost$. Then the mappings are as follows:

$$\begin{array}{lll} \texttt{flightFrom}(id, departure) & \text{:-} & \texttt{flight}(id, departure, arrival, cost) \\ \texttt{flightPrice}(id, cost) & \text{:-} & \texttt{flight}(id, departure, arrival, cost) \end{array}$$

Next we define the RDF graph induced by a set of mapping rules and a database.

Definition 4 (Virtual RDF Graph via R2RML Mapping). *Let $\mathcal{M}$ be a set of R2RML mappings (represented in Datalog), and I a relational database instance. Then the virtual RDF graph $\mathcal{M}(I)$ is defined as the RDF graph corresponding to the minimal model of $\mathcal{M} \cup I$, i.e., $\mathcal{M}(I) = \mathcal{G}(\mathit{MM}(\mathcal{M} \cup I))$.*

3 Answering SPARQL over Rules and Virtual RDF

In this section we describe how we translate SPARQL queries over a rule-enriched vocabulary into SQL. The translation consists of two steps: *(i)* translation of the SPARQL query and RIF rules into a recursive Datalog program, and *(ii)* generation of an SQL query (with CTEs) from the Datalog program.

3.1 SPARQL to Recursive Datalog

The translation we present here extends the one described in [18,19,23], where the authors define a translation function τ from SPARQL queries to *non-recursive* Datalog programs with stratified negation. Due to space limitations, we do not provide the details of the translation τ, but illustrate it with an example, and refer to [18,19] for its correctness. Note, in this paper we only consider BGPs corresponding to atoms in SWRL or RIF rules; in other words, triple patterns like $(t_1, ?x, t_2)$ or $(t, \texttt{rdf:type}, ?x)$ are disallowed (cf. the restrictions on RIF in Section 2.3).

Example 5. Consider the query (V, P) in Example 1, for which we report below the algebra expression in which we have labeled each sub-expression P_i of P.

```
Filter (?z < 100)(                                      # P1
                    Join(                               # P2
                       BGP(?x :tripPlanFrom :Bolzano .)  # P3
                       Join(                            # P4
                            BGP(?x :tripPlanTo ?y .)    # P5
                            BGP(?x :tripPrice  ?z .)))) # P6
```

The Datalog translation contains one predicate $\texttt{ans}_i$ representing each algebra sub-expression P_i. The Datalog program $\tau(V, P)$ for this query is as follows:

$$\begin{array}{lll}
\texttt{ans}_1(x) & \text{:-} & \texttt{ans}_2(x,y,z),\ \texttt{Filter}(z > 100) \\
\texttt{ans}_2(x,y) & \text{:-} & \texttt{ans}_3(x),\ \texttt{ans}_4(x,y,z) \\
\texttt{ans}_3(x) & \text{:-} & \texttt{tripPlanFrom}(x, \texttt{:Bolzano}) \\
\texttt{ans}_4(x,y,z) & \text{:-} & \texttt{ans}_5(x,y),\ \texttt{ans}_6(x,z) \\
\texttt{ans}_5(x,y) & \text{:-} & \texttt{tripPlanTo}(x,y) \\
\texttt{ans}_6(x,z) & \text{:-} & \texttt{tripPrice}(x,z)
\end{array}$$

The overall translation of a SPARQL query and a set of rules to recursive Datalog is defined as follows.

Definition 5. *Let $Q = (V, P)$ be a SPARQL query and Π a set of rules. We define the translation of Q and Π to Datalog as the Datalog program $\mu(Q, \Pi) = \Pi \cup \tau(V, P)$.*

Observe that while $\tau(V, P)$ is not recursive, Π and therefore $\mu(Q, \Pi)$ might be so.

Proposition 1. *Let (G, Π) be an RDF-rule combination, Q a SPARQL query, and θ a solution mapping. Then*

$$\theta \in [\![Q]\!]_{G,\Pi} \qquad \textit{if and only if} \qquad \mu(Q, \Pi) \cup \mathcal{A}(G) \models \mathtt{ans}_Q(\theta)$$

where $\mathtt{ans}_Q$ is the predicate in $\mu(Q, \Pi)$ corresponding to Q.

Proof. Let $Q = (V, P)$. By definition, $\tau(V, P)$ is a stratified Datalog program and we assume that it has a stratification $(S_0, \ldots, S_n)$. As Π is a positive program, clearly $(\Pi \cup \mathcal{A}(G), S_0, \ldots, S_n)$ is a stratification of $\mu(Q, \Pi) \cup \mathcal{A}(G)$. Then the following statements are equivalent:

- $\theta \in [\![Q]\!]_{G,\Pi} = [\![Q]\!]_{\mathcal{G}(\mathsf{MM}(\mathcal{A}(G)\cup\Pi))}$
- *(By the correctness of the translation $\tau(V, P)$ [19])*
  $\mathtt{ans}_Q(\theta) \in \mathsf{MM}(\tau(V, P) \cup \mathsf{MM}(\mathcal{A}(G) \cup \Pi))$
- *(By the definition of GL-reduction)*
  $\mathtt{ans}_Q(\theta) \in \mathsf{MM}(\tau(V, P)^{\mathsf{MM}(\mathcal{A}(G)\cup\Pi)})$
- *(By the definition of model of a stratified Datalog program)*
  $\mathtt{ans}_Q(\theta) \in \mathsf{MM}(\tau(V, P) \cup \mathcal{A}(G) \cup \Pi) = \mathsf{MM}(\mu(Q, \Pi) \cup \mathcal{A}(G))$
- $\mu(Q, \Pi) \cup \mathcal{A}(G) \models \mathtt{ans}_Q(\theta)$ □

Considering that R2RML mappings are represented in Datalog, we immediately obtain the following result.

Corollary 1. *Let $Q = (V, P)$ be a SPARQL query, Π a set of rules, $\mathcal{M}$ a set of R2RML mappings, I a database instance, and θ a solution mapping. Then*

$$\theta \in [\![Q]\!]_{\mathcal{M}(I),\Pi} \qquad \textit{if and only if} \qquad \mu(Q, \Pi) \cup \mathcal{M} \cup I \models \mathtt{ans}_Q(\theta)$$

3.2 Recursive Datalog to Recursive SQL

In this section, we show how to translate to SQL the recursive Datalog program obtained in the previous step. Note that translation from Datalog to SQL presented in [23] does not consider recursive rules. Here we extend such translation to handle recursion by making use of SQL common table expressions (CTEs). We assume that the extensional database (EDB) predicates (i.e., those not appearing in the heads of Datalog rules) are stored in the database.

Fixpoint Operator, Linear Recursive Query and CTE. We recall first how relational algebra can be extended with a fixpoint operator [1]. Consider an equation of the form $R = f(R)$, where $f(R)$ is a relational algebra expression over R. A least fixpoint of the equation above, denoted $\mathsf{LFP}(R = f(R))$, is a relation S such that

- $S = f(S)$
- if R is any relation such that $R = f(R)$, then $S \subseteq R$.

In order to ensure the existence of a fixpoint (and hence of the least fixpoint), f must be monotone.

Consider now the equation $R = f(R_1, \ldots, R_n, R)$, making use of a relational algebra expression over R and other relations. Such equation is a *linear* recursive expression, if R occurs exactly once in the expression f. Then, we can split f into a non-recursive part f_1 not mentioning R, and a recursive part f_2, i.e., $f(R_1, \ldots, R_n, R) = f_1(R_1, \ldots, R_n) \cup f_2(R_1, \ldots, R_n, R)$. In this case, $\mathsf{LFP}(R = f(R_1, \ldots, R_n, R))$ can be expressed using a *common table expression* (CTE) of SQL99:

```
WITH RECURSIVE R AS {
  [block for base case f1]
UNION
  [block for recursive case f2]
}
```

We remark that CTEs are already implemented in most of the commercial databases, e.g, Postgres, MS SQL Server, Oracle, DB2, H2, HSQL.

Example 6. Suppose we have the database relation `Flight` (f, for short), with attributes id, $source$, $destination$, and $cost$ (i, s, d, c, for short) and we want to compute all the possible routes such that the total cost is less that 100. To express this relation `plan` we can use the following equation with least fixpoint operator:

$$\texttt{plan} = \mathsf{LFP}(f^* = \pi_{var_1}(f) \cup \pi_{var_2}(\rho_{count}(\sigma_{fil}(f \bowtie f^*))))$$

where

$$var_1 = f.s, f.d, f.c \qquad count = (f.c + f^*.c)/c$$
$$var_2 = f.s, f^*.d, c \qquad fil = f.c + f^*.c < 100,\ f.s = f^*.d$$

It can be expressed as the following CTE:

```
WITH RECURSIVE plan AS (
  SELECT  f.s, f.d, f.c FROM f
UNION
  SELECT  plan.s, f.d, f.c + plan.c AS c
  FROM f, plan
  WHERE f.c + plan.c < 100 AND f.s = plan.d
)
```

Now we proceed to explain how to translate a recursive program into a fixpoint relation algebra expression.

The *dependency graph* for a Datalog program is a directed graph representing the relation between the predicate names in a program. The set of nodes are the relation symbols in the program. There is an arc from a node a to a node b if and only if a appears in the body of a rule in which b appears in the head. A program is *recursive* if there is a cycle in the dependency graph.

We say that a Datalog program Π is SQL99 compatible if *(i)* there are no cycles in the dependency graph of Π apart from self-loops; and *(ii)* the recursive predicates are restricted to linear recursive ones. For the sake of simplicity, and ease the presentation, we assume that non-recursive rules have the form

$$p(\boldsymbol{x}) \text{ :- } atom_1(\boldsymbol{x}_1), \ldots, atom_n(\boldsymbol{x}_n), cond(\boldsymbol{z}) \tag{1}$$

where each $atom_i$ is a relational atom, and $cond(\boldsymbol{z})$ is a conjunction of atoms over built-in predicates. Moreover, for each recursive predicate p, there is a pair of rules defining p of the form

$$\begin{array}{lcl} p(\boldsymbol{x}) & \text{:-} & atom_0(\boldsymbol{x}_0), p(\boldsymbol{y}), cond_1(\boldsymbol{z}_1) \\ p(\boldsymbol{x}) & \text{:-} & atom_1(\boldsymbol{x}_1), \ldots, atom_n(\boldsymbol{x}_n), cond_2(\boldsymbol{z}_2) \end{array} \tag{2}$$

In addition, we assume that equalities between variables in the rules are made explicit in atoms of the form $x = y$. Thus, there are no repeated variables in non-equality atoms.

The intuition behind the next definition is that for each predicate p of arity n in the program, we create a relational algebra expression $\mathsf{RA}(p)$.

Definition 6. *Let p be a predicate and Π a set of Datalog rules.*

- *If p is an extensional predicate, then*

$$\mathit{RA}(p(\boldsymbol{x})) = p$$

- *If p is a non-recursive intensional predicate, let Π_p be the set of rules in Π defining p. For such a rule r, which is of the form* (1)*, let*

$$\mathit{RA}(r) = \sigma_{cond}(\mathit{RA}(atom_1(\boldsymbol{x}_1)) \bowtie \cdots \bowtie \mathit{RA}(atom_n(\boldsymbol{x}_n)))$$

  where $cond$ is the condition corresponding to the conjunction of atoms $cond(\boldsymbol{z})$ in (1)*. Then*

$$\mathit{RA}(p(\boldsymbol{x})) = \bigcup_{r \in \Pi_p} \mathit{RA}(r)$$

- *If p is a recursive intensional predicate defined by a pair of rules of the form* (2)*, then*

$$\begin{array}{rl} \mathit{RA}(p(\boldsymbol{x})) = \mathit{LFP}(p = & \sigma_{cond_1}(\mathit{RA}(atom_0(\boldsymbol{x}_0)) \bowtie p) \cup \\ & \sigma_{cond_2}(\mathit{RA}(atom_1(\boldsymbol{x}_1)) \bowtie \cdots \bowtie \mathit{RA}(atom_n(\boldsymbol{x}_n)))) \end{array}$$

  where again $cond_1$ and $cond_2$ are the conditions corresponding to the conjunctions of atoms $cond_1(\boldsymbol{z}_1)$ and $cond_2(\boldsymbol{z}_2)$ in the two rules defining p.

The next proposition shows that if the rule component of an RDF-rule combination is SQL99 compatible, then the Datalog transformation of the combination is also SQL99 compatible.

Proposition 2. *Let (G, Π) be an RDF-rule combination, $Q = (V, P)$ a SPARQL query, $\mathcal{M}$ a set of R2RML mapping. If Π is SQL99 compatible, then $\mu(Q, \Pi) \cup \mathcal{M}$ is also SQL99 compatible.*

Proof (Sketch). This can be easily verified by checking the layered structure of $\mu(Q, \Pi) \cup \mathcal{M} = \tau(V, P) \cup \Pi \cup \mathcal{M}$. Observe that (1) $\tau(V, P)$ and $\mathcal{M}$ are non-recursive Datalog programs, and (2) there is no arc from the head predicates of Π (resp. $\tau(V, P)$) to the body predicates of $\mathcal{M}$ (resp. Π) in the dependency graph of $\mu(Q, \Pi) \cup \mathcal{M}$. Therefore no additional cycles in the dependency graph will be introduced except the ones already in Π (if any).

4 Embedding Entailments in Mappings Rules

A first naive implementation of the technique described above was unable to generate SQL queries, due to the blowup caused by the processing of mappings, i.e., by *unfolding* the predicates defined in the heads of mapping rules with the corresponding queries in the bodies. Thus, it was necessary to optimize the rules and the mappings before unfolding the query. In this section, we present two optimizations based on the key observation that we can optimize the rules together with R2RML mappings independently of the SPARQL queries.

1. For the recursive rules, we can pre-compute the relational algebra expression (i.e., the recursive common table expressions (CTEs) in SQL).
2. For the non-recursive rules, we introduce a method to embed entailments into the mapping rules.

4.1 Recursion Elimination

The presence of recursion in the Datalog representation of the SPARQL query gives rise to a number of issues e.g., efficiently unfolding the program using SLD resolution, and managing the different types of variables. For this reason, before generating SQL for the whole set of rules, *(i)* we pre-compute CTEs for recursive predicates, making use of the expressions in relational algebra extended with fixpoints provided in Definition 6, *(ii)* we eliminate the recursive rules and replace the recursive predicates by fresh predicates; these fresh predicates are defined by cached CTEs.

4.2 T-Mappings for SWRL Rules

We introduce now an extension of the notion of *T-mappings* [20] to cope with SWRL rules. T-mappings have been introduced in the context of OBDA systems in which queries posed over an OWL 2 QL ontology that is mapped to a relational database, are rewritten in terms of the database relations only. They allow one to embed in the mapping assertions entailments over the data that are caused by the ontology axioms, and thus to obtain a more efficient Datalog program. In our setting, T-mappings extend the set of mapping to embed entailments caused by (recursive) rules into the mapping rules. Formally:

Definition 7 (SWRL T-Mappings). *Let $\mathcal{M}$ be a set of mappings, I a database instance, and Π a set of SWRL rules. A* T-mapping for Π w.r.t. $\mathcal{M}$ *is a set $\mathcal{M}_\Pi$ of mappings such that: (i) every triple entailed by $\mathcal{M}(I)$ is also entailed by $\mathcal{M}_\Pi(I)$; and (ii) every fact entailed by $\Pi \cup \mathcal{A}(\mathcal{M}(I))$ is also entailed by $\mathcal{M}_\Pi(I)$.*

A T-mapping for SWRL rules can be constructed iteratively, unlike OWL 2 QL-based T-mappings, using existing mappings to generate new mappings that take into account the implications of the rules[3]. In Algorithm 1, we describe the construction process which is similar to the classical semi-naive evaluation for Datalog programs.

[3] Recall that SWRL allows only for unary and binary predicates.

Algorithm 1: T-Mapping(Π,$\mathcal{M}$)

```
Input: a set $\Pi$ of (SWRL) rules; a set $\mathcal{M}$ of R2RML mappings
Output: T-Mapping $\mathcal{M}_\Pi$ of $\mathcal{M}$ w.r.t. $\Pi$
$\Delta\mathcal{M} \leftarrow \mathcal{M}$;   $\mathcal{M}_\Pi \leftarrow \emptyset$;
while $\Delta\mathcal{M} \neq \emptyset$ do
    $\mathcal{M}_\Pi \leftarrow \mathcal{M}_\Pi \cup \Delta\mathcal{M}$;   $\Delta\mathcal{M}' \leftarrow \emptyset$;
    foreach mapping $m \in \Delta\mathcal{M}$ do
        foreach rule $r \in \Pi$ do
            if $m$ and $r$ resolves then
                $\Delta\mathcal{M}' \leftarrow \Delta\mathcal{M}' \cup res(m,r)$ ;        ▷ $res(m,r)$ is a set of mappings
        $\Delta\mathcal{M} \leftarrow \Delta\mathcal{M}'$;
return $\mathcal{M}_\Pi$
```

Theorem 1. *Let $\mathcal{M}$ be a set of mappings and Π a set of SWRL rules. Then there exists always a T-mapping for Π w.r.t. $\mathcal{M}$.*

Proof (sketch). Let I be a database instance. The Datalog program $\mathcal{M} \cup \Pi \cup I$ does not contain negation, therefore, it has a unique minimal model, which can be computed via a repeated exhaustive application of the rules in it in a bottom-up fashion.

Since the rules in $\mathcal{M}$ act as links between the atoms in Π and the ground facts in I, it is clear that if a predicate A in Π does not depend on an intensional predicate in $\mathcal{M}$, every rule containing A can be removed without affecting the minimal model of $\mathcal{M} \cup \Pi \cup I$. Thus, we can safely assume that every predicate in Π depends on an intensional predicate in $\mathcal{M}$. Thus, if every predicate in a rule is defined (directly or indirectly) by mappings in $\mathcal{M}$, we can always replace the predicate by its definition, leaving in this way rules whose body uses only database relations. □

5 Implementation

The techniques presented here are implemented in the *Ontop*[4] system. *Ontop* is an open-source project released under the Apache License, developed at the Free University of Bozen-Bolzano and part of the core of the EU project Optique[5]. *Ontop* is available as a plugin for Protege 4, as a SPARQL end-point, and as OWLAPI and Sesame libraries. To the best of our knowledge, *Ontop* is the first system supporting all the following W3C recommendations: OWL 2 QL, R2RML, SPARQL, and SWRL[6]. Support for RIF and integration of SWRL and OWL 2 QL ontologies will be implemented in the near future.

In Figure 2, we depict the new architecture that modifies and extends our previous OBDA approach, by replacing OWL 2 QL with SWRL. First, during loading time, we translate the SWRL rules and the R2RML mappings into a Datalog program. This set of

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

[5] `http://www.optique-project.eu/`

[6] SWRL is a W3C submission, but not a W3C recommendation yet.

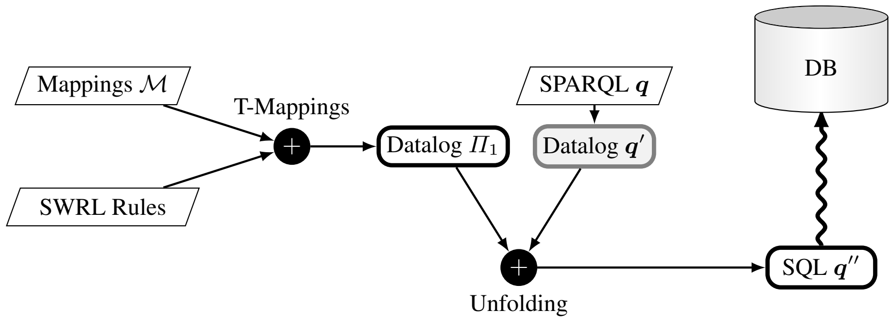

Fig. 2. SWRL and Query processing in the *Ontop* system

rules is then optimised as described in Section 4. This process is query independent and is performed only once when *Ontop* starts. Then the system translates the SPARQL query provided by the user into another Datalog program. None of these Datalog programs is meant to be executed. They are only a formal and succinct representation of the rules, the mappings, and the query, in a single language. Given these two Datalog programs, we unfold the query with respect to the rules and the mappings using SLD resolution. Once the unfolding is ready, we obtain a program whose vocabulary is contained in the vocabulary of the datasource, and therefore can be translated to SQL. The technique is able to deal with all aspects of the translation, including URI and RDF Literal construction, RDF typing, and SQL optimisation. However, the current implementation supports only a restricted form of queries involving recursion: SPARQL queries with recursion must consist of a single triple involving the recursive predicate. This preliminary implementation is meant to test performance and scalability.

6 Evaluation

To evaluate the performance and scalability of *Ontop* with SWRL ontologies, we adapted the NPD benchmark. The NPD benchmark [7] is based on the *Norwegian Petroleum Directorate*[7] *Fact Pages*, which contains information regarding the petroleum activities on the Norwegian continental shelf. We used PostgreSQL as the underlying relational database system. The hardware consisted of an HP Proliant server with 24 Intel Xeon X5690 CPUs (144 cores @3.47GHz), 106GB of RAM and a 1TB 15K RPM HD. The OS is Ubuntu 12.04 LTS.

The original benchmark comes with an OWL ontology[8]. In order to test our techniques, we translated a fragment of this ontology into SWRL rules by *(i)* converting the OWL axioms into rules whenever possible; and *(ii)* manually adding linear recursive rules. The resulting SWRL ontology contains 343 concepts, 142 object properties, 238 data properties, 1428 non-recursive SWRL rules, and 1 recursive rule. The R2RML file

[7] `http://www.npd.no/en/`

[8] `http://sws.ifi.uio.no/project/npd-v2/`

Table 1. Evaluation of *Ontop* on NPD benchmark

		Load	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_{10}	q_{11}	q_{12}	r_1
NPD	*Ontop*	16.6	0.1	0.09	0.03	0.2	0.02	1.7	0.1	0.07	5.6	0.1	1.4	2.8	0.25
	Stardog	-	2.06	0.65	0.29	1.26	0.20	0.34	1.54	0.70	0.06	0.07	0.11	0.15	-
NPD	*Ontop*	17.1	0.12	0.13	0.10	0.25	0.02	3.0	0.2	0.2	5.7	0.3	6.7	8.3	27.8
($\times$2)	Stardog	-	5.60	1.23	0.85	1.89	0.39	2.29	2.41	1.47	0.34	0.36	1.78	1.52	-
NPD	*Ontop*	16.7	0.2	0.3	0.17	0.67	0.05	18.08	0.74	0.35	6.91	0.55	162.3	455.4	237.6
($\times$10)	Stardog	-	8.89	1.43	1.17	2.04	0.51	4.12	5.84	5.30	0.42	0.72	3.03	3.86	–

includes 1190 mappings. The NPD query set contains 12 queries obtained by interviewing users of the NPD data.

We compared *Ontop* with the only other system (to the best of our knowledge) offering SWRL reasoning over on-disk RDF/OWL storage : *Stardog 2.1.3*. Stardog[9] is a commercial RDF database developed by Clark&Parsia that supports SPARQL 1.1 queries and OWL 2/SWRL for reasoning. Since Stardog is a triple store, we needed to materialize the virtual RDF graph exposed by the mappings and the database using *Ontop*. In order to test the scalability w.r.t. the growth of the database, we used the data generator described in [7] and produced several databases, the largest being approximately 10 times bigger than the original NPD database. The materialization of NPD (x2) produced 8,485,491 RDF triples and the materialization of NPD (x10) produced 60,803,757 RDF triples. The loading of the last set of triples took around one hour.

The results of the evaluation (in seconds) are shown in Table 1. For queries q_1 to q_{12}, we only used the non-recursive rules and compared the performance with Stardog. For the second group (r_1), we included recursive rules, which can only be handled by *Ontop*.

Discussion. The experiments show that the performance obtained with *Ontop* is comparable with that of Stardog and in most queries *Ontop* is faster. There are 4 queries where *Ontop* performs poorly compared to Stardog. Due to space limitations, we will analyze only one of these 4; however, the reason is the same in each case. Consider query 12, which is a complex query that produces an SQL query with 48 unions. The explosion in the size of the query is produced by interaction of long hierarchies below the concepts used in the query and multiple mappings for each of these concepts. For instance `npdv:Wellbore` has 24 subclasses, and `npdv:name` has 27 mappings defining it. Usually just a join between these two should generate a union of $24\times27 = 648$ SQL queries. *Ontop* manages to optimize this down to 48 unions but more work needs to be done to get better performance. This problem is not a consequence of the presence of rules, but is in the very nature of the OBDA approach, and is one of the main issues to be studied in the future. For Stardog, the situation is slightly easier as it works on the RDF triples directly and does not need to consider mappings.

[9] `http://stardog.com/`

7 Conclusion

In this paper we have studied the problem of SPARQL query answering in OBDA in the presence of rule-based ontologies. We tackle the problem by rewriting the SPARQL queries into recursive SQLs. To this end we provided a translation from SWRL rules into relational algebra extended with fixed-point operators that can be expressed in SQL99's Common Table Expressions (CTEs). We extended the existing T-mapping optimisation technique in OBDA, proved that for every non-recursive SWRL program there is a T-mapping, and showed how to construct it. The techniques presented in this paper were implemented in the system *Ontop*. We evaluated its scalability and compared the performance with the commercial triple store Stardog. Result shows that most of the SQL queries produced by *Ontop* are of high quality, allowing fast query answering even in the presence of big data sets and complex queries.

Acknowledgement. This paper is supported by the EU under the large-scale integrating project (IP) Optique (Scalable End-user Access to Big Data), grant agreement n. FP7-318338. We thank Héctor Pérez-Urbina for his support in evaluating Stardog.

References

1. Aho, A.V., Ullman, J.D.: The universality of data retrieval languages. In: Proc. of the 6th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL 1979). pp. 110–120 (1979)
2. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The *DL-Lite* family and relations. J. of Artificial Intelligence Research 36, 1–69 (2009)
3. Bienvenu, M., Ortiz, M., Simkus, M., Xiao, G.: Tractable queries for lightweight description logics. In: Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI 2013). IJCAI/AAAI (2013)
4. Boley, H., Kifer, M.: A guide to the basic logic dialect for rule interchange on the Web. IEEE Trans. on Knowledge and Data Engineering 22(11), 1593–1608 (2010)
5. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodríguez-Muro, M., Rosati, R.: Ontologies and databases: The *DL-Lite* approach. In: Tessaris, S., Franconi, E. (eds.) Reasoning Web. Semantic Technologies for Informations Systems – 5th Int. Summer School Tutorial Lectures (RW 2009), Lecture Notes in Computer Science, vol. 5689, pp. 255–356. Springer (2009)
6. 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)
7. Calvanese, D., Lanti, D., Rezk, M., Slusnys, M., Xiao, G.: Data generation for OBDA systems benchmarks. In: Proc. of The 3rd OWL Reasoner Evaluation Workshop (ORE 2014). CEUR-WS.org (2014)
8. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF mapping language. W3C Recommendation, World Wide Web Consortium (Sep 2012), available at `http://www.w3.org/TR/r2rml/`
9. Eiter, T., Ortiz, M., Simkus, M., Tran, T.K., Xiao, G.: Query rewriting for Horn-SHIQ plus rules. In: Proc. of the 26th AAAI Conf. on Artificial Intelligence (AAAI 2012). AAAI Press (2012)

10. Glimm, B., Ogbuji, C.: SPARQL 1.1 Entailment Regimes. W3C Recommendation, World Wide Web Consortium (Mar 2013), available at `http://www.w3.org/TR/sparql11-entailment/`
11. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language. W3C Recommendation, World Wide Web Consortium (Mar 2013), available at `http://www.w3.org/TR/sparql11-query`
12. Kifer, M., Boley, H.: RIF Overview (Second Edition). W3C working group note 5 February 2013, World Wide Web Consortium (2013), available at `http://www.w3.org/TR/2013/NOTE-rif-overview-20130205/`
13. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The combined approach to ontology-based data access. In: Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI 2011). pp. 2656–2661 (2011)
14. Lloyd, J.W.: Foundations of Logic Programming (Second, Extended Edition). Springer, Berlin, Heidelberg (1987)
15. Manola, F., Mille, E.: RDF primer. W3C Recommendation, World Wide Web Consortium (Feb 2004), available at `http://www.w3.org/TR/rdf-primer-20040210/`
16. Pérez-Urbina, H., Motik, B., Horrocks, I.: Tractable query answering and rewriting under description logic constraints. J. of Applied Logic 8(2), 186–209 (2010)
17. 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)
18. Polleres, A.: From SPARQL to rules (and back). In: Proc. of the 16th Int. World Wide Web Conf. (WWW 2007). pp. 787–796 (2007)
19. 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)
20. Rodríguez-Muro, M., Calvanese, D.: Dependencies: Making ontology based data access work in practice. In: Proc. of the 5th Alberto Mendelzon Int. Workshop on Foundations of Data Management (AMW 2011). CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/`, vol. 749 (2011)
21. Rodriguez-Muro, M., Calvanese, D.: High performance query answering over *DL-Lite* ontologies. In: Proc. of the 13th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2012). pp. 308–318 (2012)
22. Rodriguez-Muro, M., Kontchakov, R., Zakharyaschev, M.: Ontology-based data access: Ontop of databases. In: Proc. of the 12th Int. Semantic Web Conf. (ISWC 2013). Lecture Notes in Computer Science, vol. 8218, pp. 558–573. Springer (2013)
23. Rodriguez-Muro, M., Rezk, M.: Efficient SPARQL-to-SQL with R2RML mappings. Tech. rep., Free University of Bozen-Bolzano (Jan 2014), available at `http://www.inf.unibz.it/~mrezk/pdf/sparql-sql.pdf`