Applications and Practices in Ontology Design, Extraction, and Reasoning
G. Cota et al. (Eds.)
IOS Press, 2020
© 2020 Akademische Verlagsgesellschaft AKA GmbH, Berlin. All rights reserved.
doi:10.3233/SSW200044

Querying the Semantic Web via Rules

Marcelo ARENAS [a], Georg GOTTLOB [b], Andreas PIERIS [c]
[a] *Universidad Católica & IMFD Chile, Chile, e-mail: marenas@ing.puc.cl*
[b] *University of Oxford, UK & TU Wien, Austria e-mail: georg.gottlob@cs.ox.ac.uk*
[c] *University of Edinburgh, UK, e-mail: apieris@inf.ed.ac.uk*

Abstract. The problem of querying RDF data is a central issue for the development of the Semantic Web. The query language SPARQL has become the standard language for querying RDF since its W3C standardization in 2008. However, the 2008 version of this language missed some important functionalities: reasoning capabilities to deal with RDFS and OWL vocabularies, navigational capabilities to exploit the graph structure of RDF data, and a general form of recursion much needed to express some natural queries. To overcome those limitations, a new version of SPARQL, called SPARQL 1.1, was released in 2013, which includes entailment regimes for RDFS and OWL vocabularies, and a mechanism to express navigation patterns through regular expressions. Nevertheless, there are useful navigation patterns that cannot be expressed in SPARQL 1.1, and the language lacks a general mechanism to express recursive queries. This chapter is a gentle introduction to a tractable rule-based query language, in fact, an extension of Datalog with value invention, stratified negation, and falsum, that is powerful enough to define SPARQL queries enhanced with the desired functionalities focussing on a core fragment of the OWL 2 QL profile of OWL 2.

Keywords. RDF, SPARQL, Querying RDF

1. Introduction

The Resource Description Framework (RDF) is the W3C recommendation data model to represent information about World Wide Web resources. An atomic piece of data in RDF is a *Uniform Resource Identifier (URI)*. In the RDF data model, URIs are organized as RDF graphs, that is, labeled directed graphs where node labels and edge labels are URIs. As with any other data model, the problem of querying RDF data has been widely studied. Since its release in 1998, several designs and implementations of RDF query languages have been proposed [16]. In 2004, a first public working draft of a language, called SPARQL, was released by the W3C, which is in fact a graph-matching query language. Since then, SPARQL has been adopted as the standard language for querying the Semantic Web, and in 2008 it became a W3C recommendation.[1]

One of the distinctive features of Semantic Web data is the existence of vocabularies with predefined semantics: the *RDF Schema (RDFS)*[2] and the *Web Ontology Language (OWL)*[3], which can be used to derive logical conclusions from RDF graphs. Moreover, it

[1] http://www.w3.org/TR/rdf-sparql-query
[2] http://www.w3.org/TR/rdf-schema
[3] http://www.w3.org/TR/owl-features/

has been recognized that navigational capabilities are of fundamental importance for data models with an explicit graph structure such as RDF [2,6,8,15,27], and, more generally, it is well-accepted that a general form of recursion is a central feature for a graph query language [8,21,31]. Therefore, it would be desirable to have an RDF query language equipped with reasoning capabilities to deal with the RDFS and OWL vocabularies, as well as a general mechanism to express recursive queries. Unfortunately, the 2008 version of SPARQL missed the above crucial functionalities. To overcome these limitations, a new version, called SPARQL 1.1 [19], was released in 2013, which includes entailment regimes for RDFS and OWL vocabularies, and a mechanism to express navigation patterns through regular expressions. However, it has already been observed that there exist some very natural queries that require a more general form of recursion and cannot be expressed in SPARQL 1.1 [21,31].

Towards an Expressive RDF Query Language. Datalog with stratified negation [1,14] has been shown to be expressive enough to represent every SPARQL query [2,3,6,28,32]. Thus, it has been used as a natural platform for SPARQL extensions with richer navigation capabilities and recursion mechanisms [21,31]. Moreover, some extensions of Datalog with existential quantification in rule-heads are appropriate to encode some inferencing mechanisms in OWL [11]. Therefore, as discussed in [5,18], Datalog and some of its extensions (in particular, the members of the Datalog$^{\pm}$ family of knowledge representation and query languages [12]) appear to be suitable for bridging the gap between RDF query languages and the desired functionalities, that is, reasoning capabilities and a general mechanism to express recursive queries. However, query answering under the language obtained by extending Datalog with existential quantification is undecidable; this is implicit in [9]. In fact, the undecidability holds even in the case of data complexity, i.e., when the input query is fixed, and only the extensional database (or the RDF graph) is considered as part of the input [11]. It is thus a non-trivial task to isolate an expressive RDF query language that

1. is based on Datalog, which enables a modular rule-based style of writing queries;
2. is expressive enough for being useful in practice, and, in particular, to support reasoning and navigational capabilities, as well as a general form of recursion;
3. ensures the decidability of the query evaluation problem; and
4. has good complexity properties in the case the input query is fixed – this is of fundamental importance as a low data complexity is considered to be a key condition for a query language to be useful in practice.

Triple Query Language. A first attempt to design a Datalog-based RDF query language that fulfills the above desiderata, focussing on the profile OWL 2 QL of OWL 2, was made in [4]. The proposed language, called TriQ-Lite,[4] is based on Datalog$^{\exists,\neg s,\bot}$, that is, Datalog extended with existential quantification in rule-heads, stratified negation, and negative constraints expressed by using the symbol $\bot$ (falsum) in rule-heads. However, TriQ-Lite suffers from a serious drawback, which may revoke its advantage as an expres-

[4]This language is the lite version of a highly expressive language called TriQ, which stands for triple query language, also introduced in [4].

sive RDF query language, namely it is not a plain language. We call a rule-based query language plain if it allows the user to express a query as a single program in a simple non-composite syntax. An example of a plain query language is Datalog itself, where the user simply needs to define a single Datalog program that captures the intended query. The property of plainness provides conceptual simplicity, which is considered to be a key condition for a query language to be useful in practice. Although TriQ-Lite is based on an extension of Datalog, the way its syntax and semantics are defined significantly deviates from the standard way of defining Datalog-like languages, and thus does not inherit the plainness of Datalog. TriQ-Lite is a composite language, where the user is forced to split the query program in several programs $\Pi_1, \ldots, \Pi_n$ so that each Π_i can be expressed by the fragment of Datalog$^{\exists,\neg s,\bot}$ underlying TriQ-Lite, while each pair (Π_i, Π_{i+1}) is bridged via a set Q_i of conjunctive queries.

In view of the conceptual weakness of TriQ-Lite discussed above, the new version of it, dubbed TriQ-Lite 1.0, was introduced in [5,18]. TriQ-Lite 1.0 is indeed a plain language based on Datalog$^{\exists,\neg s,\bot}$ that fulfills all the crucial desiderata discussed above. The goal of this chapter is to provide a gentle introduction to TriQ-Lite 1.0 by intuitively explaining its key ingredients, without delving into the low level formal definitions and technical details. For further details we refer the interested reader to [5].

2. Motivating Scenarios and Queries

Let us first expose some of the difficulties and limitations encountered when querying RDF data with SPARQL, which further motivated the design of a Datalog-like RDF query language. The following scenarios and queries are taken from [5].

Assume that G_1 is an RDF graph consisting of:

$$(\text{dbUllman},\ \text{is\_author\_of},\ \text{“The Complete Book”}),$$
$$(\text{dbUllman},\ \text{name},\ \text{“Jeffrey Ullman”}).$$

The first triple states that the object with URI dbUllman is one of the authors of the book “The Complete Book”, while the second triple indicates that the name of dbUllman is “Jeffrey Ullman”. To retrieve the list of authors in G_1 we can use the SPARQL query:

```
SELECT ?X
WHERE {
        ?Y is_author_of ?Z .                    (1)
        ?Y name ?X }
```

Note that variables start with the symbol ?. Moreover, the expression $?Y$ is_author_of $?Z$ represents a triple pattern that is used to retrieve the pairs (a, b) of elements from G_1, which are stored in the variables $?Y$ and $?Z$, such that a is an author of b. In the same way, the expression $?Y$ name $?X$ also represents a triple pattern that is used to retrieve the pairs (a, c) of elements from G_1, which are stored in the variables $?Y$ and $?X$, such that c is the name of a. Finally, the symbol . (dot) is used as a separator of the triple patterns, whose results have to be joined when computing the answer to the query, and SELECT $?X$ indicates that we are interested in the values stored in $?X$.

In TriQ-Lite 1.0 it is assumed that a predicate triple$(\cdot,\cdot,\cdot)$ is used to store the triples of an RDF graph. Thus, query (1) can be formulated in TriQ-Lite 1.0 as follows:

$$\text{triple}(y, \text{is\_author\_of}, z), \text{triple}(y, \text{name}, x) \rightarrow \text{query}(x). \tag{2}$$

The possibility of returning an RDF graph as the answer to a SPARQL query is considered as a fundamental feature [19,29]. For this reason, one can use the CONSTRUCT operator in order to produce an RDF graph as the output of a query. For example, the following query constructs an RDF graph consisting of triples $(a, \text{name\_author}, b)$, where a is the name of an author of b:

```
CONSTRUCT { ?X name_author ?Z }
WHERE {
      ?Y is_author_of ?Z .
      ?Y name ?X }
```

The expression $?X$ name_author $?Z$ represents a triple pattern specifying which RDF triples are to be included in the output. The result of evaluating this query over G_1 is

$$(\text{“Jeffrey Ullman”},\ \text{name\_author},\ \text{“The Complete Book”}).$$

In TriQ-Lite 1.0, the user is not forced to learn about a new operator in order to produce an RDF graph as output. (S)he can simply replace in (2) the predicate query$(\cdot)$ by the predicate triple$(\cdot,\cdot,\cdot)$ in order to produce an RDF graph:

$$\text{triple}(y, \text{is\_author\_of}, z), \text{triple}(y, \text{name}, x) \rightarrow \text{triple}(x, \text{name\_author}, z). \tag{3}$$

The use of the operator CONSTRUCT in SPARQL allows to have compositionality; the output of a query can be used as the input of another query. This is a key property, which plays a crucial role when adding a recursion mechanism to SPARQL [30]. Notice that TriQ-Lite 1.0 inherits the compositionality of Datalog, so that a recursion mechanism can be introduced without needing additional syntactic constructs.

Assume now that G_2 is an RDF graph extending G_1 with the following triples:

$$(\text{dbAho},\ \text{is\_coauthor\_of},\ \text{dbUllman}),$$
$$(\text{dbAho},\ \text{name},\ \text{“Alfred Aho”}).$$

The query language SPARQL allows the use of blank nodes in the CONSTRUCT operator to include some anonymous resources in an RDF graph. For example, a blank node is used in the following query to indicate that if a is a co-author of b, then there must be some publication c such that a and b are both authors of c.

```
CONSTRUCT { ?X is_author_of _:B . ?Y is_author_of _:B }
WHERE { ?X is_coauthor_of ?Y }                                (4)
```

In the above query, $\_ : B$ is a blank node, while the expressions $?X$ is_author_of $\_ : B$ and $?Y$ is_author_of $\_ : B$ specify the triples to be constructed for every possible match of the variables $?X$ and $?Y$. The semantics of SPARQL imposes the restriction that a fresh blank node has to be used for each match of the variables $?X$ and $?Y$. Although this constraint is natural in this case, this is yet another feature of SPARQL that the user needs to remember when formulating a query. In the case of TriQ-Lite 1.0, no extra notation for the creation of anonymous resources is needed, as it allows existentially quantified variables to appear in rule-heads:

$$\text{triple}(x, \text{is\_coauthor\_of}, y) \rightarrow \exists z\, \text{triple}(x, \text{is\_author\_of}, z), \text{triple}(y, \text{is\_author\_of}, z).$$

Moreover, TriQ-Lite 1.0 can be used to lift the restriction that blank nodes are used only locally. For example, it can be used to anonymize the subjects of the triples in an RDF graph, by replacing every URI in the subject position of a triple by a blank node:

$$\begin{aligned} \text{triple}(x, y, z) &\rightarrow \text{subject}(x) \\ \text{subject}(x) &\rightarrow \exists y\, \text{bnode}(x, y) \\ \text{triple}(x, y, z), \text{bnode}(x, u) &\rightarrow \text{output}(u, y, z). \end{aligned}$$

The first rule is used to store in the predicate subject$(\cdot)$ the URIs mentioned in the subject of the triples of an RDF graph. The second rule creates a blank node for every URI in the predicate subject$(\cdot)$, which is stored in the predicate bnode$(\cdot, \cdot)$. Finally, the third rule replaces in the predicate triple$(\cdot, \cdot, \cdot)$ every URI in the subject position by its associated blank node, producing an RDF graph in the predicate output$(\cdot, \cdot, \cdot)$. The ability to anonymize the subjects of an RDF graph is a useful feature as it can allow publishing data without leaking sensitive information. It is important to note that such a query cannot be expressed by using the local semantics of blank nodes in the CONSTRUCT operator of SPARQL, as the same blank node identifying a specific resource in an RDF graph has to be used every time this resource is considered in the result of the query.

Query (4) encodes some prior knowledge about the co-authorship relation. This type of knowledge can be explicitly encoded in an RDF graph by using the RDFS and OWL vocabularies. As an example of this, assume that G_3 is an RDF graph extending G_2 with the following triples:

$$\begin{aligned} (\text{r}_1,\ \text{rdf:type},\ \text{owl:Restriction}),&\ (\text{r}_2,\ \text{rdf:type},\ \text{owl:Restriction}), \\ (\text{r}_1,\ \text{owl:onProperty},\ \text{is\_coauthor\_of}),&\ (\text{r}_2,\ \text{owl:onProperty},\ \text{is\_author\_of}), \\ (\text{r}_1,\ \text{owl:someValuesFrom},\ \text{owl:Thing}),&\ (\text{r}_2,\ \text{owl:someValuesFrom},\ \text{owl:Thing}), \\ &\ (\text{r}_1,\ \text{rdfs:subClassOf},\ \text{r}_2). \end{aligned}$$

In G_3, the URIs with prefix rdfs: are part of the RDFS vocabulary, while the URIs with prefix owl: are part of the OWL vocabulary. The first three triples of G_3 define r_1 as the class of URIs a for which there exists a URI b such that $(a, \text{is\_coauthor\_of}, b)$ holds, while the following three triples of this graph define r_2 as the class of URIs a for which there exists a URI b such that the triple $(a, \text{is\_author\_of}, b)$ holds. Finally, the last triple of G_3 indicates that r_1 is a subclass of r_2.

The above set of triples states that for every a and b such that $(a, \text{is\_coauthor\_of}, b)$ holds, it must be the case that a is an author of some publication. Thus, if we want to retrieve the list of authors mentioned in G_3, then we expect to find dbAho in this list. However, the answer to the SPARQL query (1) over G_3 does not include this URI, and we are forced to encode the semantics of the RDFS and OWL vocabularies in the query. In fact, even if we try to obtain the right answer by using SPARQL 1.1 under the entailment regimes for these vocabularies, we are forced by the restrictions of the language [17] to use a query of the form:

```
SELECT ?X
WHERE {
    ?Y name ?X .
    ?Y rdf:type ?Z .
    ?Z rdf:type owl:Restriction .
    ?Z owl:onProperty is_author_of .
    ?Z owl:someValuesFrom owl:Thing }
```

This query is obtained from (1) by replacing the expression $?Y$ is_author_of $?Z$ with the last four triples above, which explicitly state that we are looking for the objects that are authors of some publication (that is, the objects of type r_2). It is clear that the resulting query is quite complex. In TriQ-Lite 1.0 such complications can be avoided by using rules encoding the semantics of the RDFS and OWL vocabularies. For example, the following rule specifies the semantics of the owl:onProperty primitive of OWL:

$$\begin{aligned}&\text{triple}(x, \text{rdf:type}, y),\\&\text{triple}(y, \text{rdf:type}, \text{owl:Restriction}),\\&\text{triple}(y, \text{owl:onProperty}, z),\\&\text{triple}(y, \text{owl:someValuesFrom}, u) \;\rightarrow\; \exists w\, \text{triple}(x, z, w).\end{aligned}$$

Notice that a fixed set of rules is used to encode the semantics of the RDFS and OWL vocabularies. If such rules are available as a library, then the user just has to include them in order to answer queries, without needing to have prior knowledge about the semantics and inference rules for the respective vocabulary. For example, if these rules have been included, then to retrieve the list of authors mentioned in G_3 we can use query (1) again, as initially expected.

3. The Language TriQ-Lite 1.0 in a Nutshell

We now proceed to give a gentle introduction to the language TriQ-Lite 1.0. But first we briefly recall plain Datalog via the standard graph reachability example.

Plain Datalog. Datalog is a prominent query language that essentially adds recursion to the select-project-join-union fragment of relational algebra. For example, using Datalog we can compute the transitive closure of a graph, which is an inherently recursive query. In particular, assuming that $G = (V, E)$ is stored in a relational database[5]

$$D_G \; = \; \{\mathrm{edge}(a,b) \mid (a,b) \in E\},$$

we can inductively compute, starting from D_G, the transitive closure of G, which will be stored in a binary relation called answer, via the following set of Datalog rules:

$$\begin{aligned} \mathrm{edge}(x,y) &\to \mathrm{reachable}(x,y) \\ \mathrm{reachable}(x,z), \mathrm{edge}(z,y) &\to \mathrm{reachable}(x,y) \\ \mathrm{reachable}(x,y) &\to \mathrm{answer}(x,y). \end{aligned}$$

The first rule, which is essentially the base step of the inductive definition, simply states that if there exists an edge from x to y, then y is reachable from x. The second rule, which corresponds to the inductive step of the definition, states that if z is reachable from x and there is an edge from z to y, then y is reachable from x. Notice that the second rule is recursive in the sense that the definition of reachable depends on itself. Finally, the last rule computes the relation answer by simply copying the relation reachable. Eventually, the Datalog query that computes the transitive closure of a graph is the pair (Π, answer), where Π is the set of Datalog rules given above. Such a query states the following: execute the Datalog program Π on the input database D, which will basically compute a new database $\Pi(D)$ that contains D, and then return as an answer the set of tuples $\{\bar{c} \mid \mathrm{answer}(\bar{c}) \in \Pi(D)\}$.

Adding the Features $\exists$, $\neg s$ and $\bot$. The query language TriQ-Lite 1.0 is based on an extension of Datalog, and in particular on Datalog$^{\exists,\neg s,\bot}$ that extends Datalog with

- existentially quantified variables ($\exists$), i.e., variables that appear in the head but not in the body of a rule, and allow us to infer new objects that are not in the database;
- stratified negation ($\neg s$), which, as customary in the Datalog literature, is interpreted as negation as failure, i.e., $\neg R(c_1, \ldots, c_n)$ holds if we fail to derive $R(c_1, \ldots, c_n)$;
- the falsum ($\bot$), which allows us to raise inconsistency.

A Datalog$^{\exists,\neg s,\bot}$ query is a pair (Π, answer), and its evaluation over an extensional database D follows the same approach as the evaluation of a plain Datalog query described above, i.e., it relies on the execution of the Datalog$^{\exists,\neg s,\bot}$ program Π over D, which results to a new database $\Pi(D)$. However, due to the features $\exists$ and $\bot$, $\Pi(D)$ may be infinite or undefined. Consider, for example, the program Π consisting of

$$R(x,y) \;\to\; \exists z\, R(y,z) \qquad R(x,y), P(x,z) \;\to\; \bot$$

Given the database $D = \{R(a,b)\}$, it is easy to verify that $\Pi(D)$ is the infinite instance

$$\{R(a,b), R(b,\nu_1), R(\nu_1,\nu_2), R(\nu_2,\nu_3), \ldots\},$$

[5]We see a database as a finite set of relational atoms of the form $R(c_1, \ldots, c_n)$, where R is an n-ary relation, and $c_1, \ldots, c_n$ are constant values drawn from a countably infinite domain.

where $\nu_1, \nu_2, \ldots$ are new labeled null values that are used as witnesses for the existentially quantified variable z. On the other hand, for the database $D = \{R(a,b), P(b,c)\}$, $\Pi(D)$ is undefined, written as $\Pi(D) = \bot$, since the atom $R(b, \nu_1)$ obtained by executing the first rule due to $R(a,b)$, together with the database atom $P(b,a)$, trigger the body of the second rule, and the falsum is entailed, i.e., we get an inconsistency.

From the above discussion, it is clear that, given a database D and a Datalog$^{\exists,\neg s,\bot}$ query $Q = (\Pi, \text{answer})$, computing the database $\Pi(D)$, and then returning the set of tuples $\{\bar{c} \mid \text{answer}(\bar{c}) \in \Pi(D)\}$, providing that $\Pi(D)$ is defined, is not an effective procedure for evaluating Q over D since $\Pi(D)$ may be infinite. In fact, we know that the decision version of the above problem, that is, given a database D, a Datalog$^{\exists,\neg s,\bot}$ query (Π, answer), and a candidate answer $\bar{c}$, decide whether $\Pi(D) \neq \bot$ implies $\text{answer}(\bar{c}) \in \Pi(D)$, is an undecidable problem. This is implicit in [9], while the undecidability holds even in the case of data complexity, i.e., when the input query is fixed, and only the database and the candidate answer are part of the input [11]. Actually, as one might expect, the problematic feature that leads to the undecidability of query evaluation is the existentially quantified variables in rule heads. Indeed, the undecidability holds already for Datalog$^{\exists}$, whereas Datalog$^{\neg s,\bot}$ is a decidable query language.

TriQ-Lite 1.0 and Wardedness. A TriQ-Lite 1.0 query is essentially a Datalog$^{\exists,\neg s,\bot}$ query (Π, answer) such that the positive existential part of Π enjoys a syntactic condition known in the literature as wardedness [5]. In other words, (Π, answer) is a TriQ-Lite 1.0 query if the set of Datalog$^{\exists}$ rules obtained from Π by dropping all the rules with the symbol $\bot$ in the head, as well as all the negated atoms occurring in rule bodies, enjoys the wardedness condition. This leads to a well-behaved Datalog$^{\exists,\neg s,\bot}$-based query language that can serve as an expressive RDF query language that combines all the desired characteristics that have been discussed in Section 1.

The wardedness condition applies a syntactic restriction on how certain "dangerous" variables of a Datalog$^{\exists}$ program Π are used. These are body variables that can be unified with a labeled null value during the execution of Π over an extensional database, and that are also propagated to the head. For example, given the Datalog$^{\exists}$ rules

$$P(x) \rightarrow \exists z\, R(x,z) \qquad \text{and} \qquad R(x,y) \rightarrow P(y)$$

the variable y in the body of the second rule is dangerous. Indeed, once we apply the first rule, an atom of the form $R(\_, \nu)$ is generated, where ν is a null value, and then the second rule is executed with the variable y being unified with ν that is propagated to the obtained atom $P(\nu)$. It has been observed that the liberal use of dangerous variables leads to a prohibitively high computational complexity, or even undecidability, of query evaluation [11]. The main goal of wardedness is to limit the use of dangerous variables with the aim of taming the way that null values are propagated during the execution of the Datalog$^{\exists}$ program. This is achieved by posing the following two conditions:

1. the dangerous variables appear together in a single body atom α, called a ward, and
2. the atom α can share only harmless variables with the rest of the body, that is, variables that unify only with constants.

Let us conclude this section by stressing that if we drop the second condition above, then we get a syntactic condition over Datalog$^\exists$ programs known in the literature as weakly-frontier-guardedness [7], which in turn leads to the powerful language TriQ 1.0. Although query evaluation for TriQ 1.0 is decidable, it is prohibitively expensive, that is, EXPTIME-complete in data complexity [5]. It turned out that TriQ-Lite 1.0 is a "nearly" maximal tractable sublanguage of TriQ 1.0 in the sense that the mildest relaxation of wardedness that one can think of, namely at most one occurrence of exactly one harmful variable that occurs in the ward can appear also outside the ward, leads to a language for which query evaluation is EXPTIME-hard in data complexity [5]. This is a strong indication that there is no obvious way to extend wardedness, and thus, TriQ-Lite 1.0, without losing tractability in data complexity.

The question that comes up is whether TriQ-Lite 1.0 is indeed expressive enough to serve as an RDF query language that combines all the desired functionalities that have been discussed in Section 1. This is the subject of the next section.

4. From SPARQL over OWL 2 QL to TriQ-Lite 1.0

The first version of the Web ontology language OWL was released in 2004 [22]. The second version of this language, which is called OWL 2, was released in 2012 [33]. OWL 2 includes three profiles that can be implemented more efficiently [23]. One of those profiles, called OWL 2 QL, is based on the description logic DL-Lite$_\mathcal{R}$ [13] and designed to be used in applications where query answering is the most important reasoning task. As the main goal of TriQ-Lite 1.0 is to provide a rule-based query language that naturally embeds the fundamental features for querying RDF, we focus on a core fragment of OWL 2 QL, called OWL 2 QL core, which corresponds to the description logic DL-Lite$_\mathcal{R}$, and discuss that every SPARQL query under the OWL 2 QL core direct semantics entailment regime, which is inherited from the OWL 2 direct semantics entailment regime [17,20], can be naturally translated into a TriQ-Lite 1.0 query.[6] For the sake of presentation, we first omit the direct semantics entailment regime, and explain in Section 4.1 how a SPARQL query can be translated into a Datalog$^{\neg s}$ query. We then discuss in Section 4.2 how every SPARQL query under the OWL 2 QL core direct semantics entailment regime can be transformed into a TriQ-Lite 1.0 query. Moreover, we discuss in Section 4.3 that the use of TriQ-Lite 1.0 allows us to formulate SPARQL queries in a simpler way as a more natural notion of entailment, obtained by removing a restriction from the regime proposed in [17], can be easily encoded in TriQ-Lite 1.0.

4.1. Translating SPARQL into Datalog$^{\neg s}$

We explain via some illustrative examples how a SPARQL query can be translated into a Datalog$^{\neg s}$ query; the complete translation can be found in [5]. Let us clarify that we refer to the definition of SPARQL 1.0 that has been adopted in a large number of articles that formally study SPARQL under set semantics, starting from [26]. We assume that the reader is familiar with the syntax and semantics of SPARQL 1.0 as defined in [26]. In what follows, given an RDF graph G, we define the relational database

[6]We focus on OWL 2 QL core, instead of the full formalism of OWL 2 QL, for technical clarity. However, TriQ-Lite 1.0 is expressive enough to deal with all the constructs of OWL 2 QL.

$$\tau_{\text{db}}(G) = \{\text{triple}(a, b, c) \mid (a, b, c) \in G\},$$

i.e., the instance of the relational schema $\{\text{triple}(\cdot, \cdot, \cdot)\}$ naturally associated with G.

Example 1 *We give a series of graph patterns, taken from [5], where their structural complexity is progressively increased, and explain how they are encoded in* Datalog$^{\neg s}$.

- *We first consider the graph pattern*

  $$P_1 = (?X, \textit{name}, ?Y),$$

  where name is a constant, that asks for the list of pairs (a, b) of elements from an RDF graph G such that b is the name of a in G. This graph pattern can be easily represented as a Datalog program over $\tau_{\text{db}}(G)$:

  $$\text{triple}(x, \textit{name}, y) \rightarrow \text{query}_{P_1}(x, y).$$

  The predicate $\text{query}_{P_1}(\cdot, \cdot)$ *is used to store the answer to the graph pattern P_1.*

- *Now consider the graph pattern*

  $$P_2 = (?X, \textit{name}, \_\!:\!B),$$

  where $\_\!:\!B$ is a blank node. This time we are asking for the list of elements in an RDF graph G that have a name (the blank node $\_\!:\!B$ is used in P_2 to indicate that $?X$ has a name, but that we are not interested in retrieving it). As in the previous case, this graph pattern can be easily represented as a Datalog program over $\tau_{\text{db}}(G)$:

  $$\text{triple}(x, \textit{name}, y) \rightarrow \text{query}_{P_2}(x).$$

  Given that blank nodes are used as existential variables in basic graph patterns, y is used in the previous rule to represent $\_\!:\!B$. However, this time we do not include the variable y in the head of the rule as we are not interested in retrieving names.

- *As a third example, consider the graph pattern:*

  $$P_3 = \underbrace{(?X, \textit{name}, ?Y)}_{P_3^1} \ \text{OPT} \ \underbrace{(?X, \textit{phone}, ?Z)}_{P_3^2},$$

  where phone is a constant. For every constant a in an RDF graph G, this graph pattern is asking for the name and phone number of a, if the information about the phone number of a is available in G, and otherwise it is only asking for the name of a. The basic graph patterns P_3^1 and P_3^2 are represented via the rules

  $$\text{triple}(x, \textit{name}, y) \rightarrow \text{query}_{P_3^1}(x, y)$$

  $$\text{triple}(x, \textit{phone}, z) \rightarrow \text{query}_{P_3^2}(x, z).$$

  The predicates $\text{query}_{P_3^1}(\cdot, \cdot)$ *and* $\text{query}_{P_3^2}(\cdot, \cdot)$ *are used in the representation of graph pattern P_3 in* Datalog$^{\neg s}$. *More precisely, we first construct a set of rules for the cases where the information about phone numbers is available:*

$$\text{query}_{P_3^1}(x,y), \text{query}_{P_3^2}(x,z) \rightarrow \text{query}_{P_3}(x,y,z)$$
$$\text{query}_{P_3^1}(x,y), \text{query}_{P_3^2}(x,z) \rightarrow \text{compatible}_{P_3}(x).$$

As for the previous graph patterns, we use $\text{query}_{P_3}(\cdot,\cdot,\cdot)$ *to store the answers to the query. But in this case, we also include a predicate* $\text{compatible}_{P_3}(\cdot)$*, which stores the individuals with phone numbers. This is used in the definition of the third rule utilized to represent* P_3*, which takes care of the individuals without phone numbers:*

$$\text{query}_{P_3^1}(x,y), \neg\text{compatible}_{P_3}(x) \rightarrow \text{query}^{\{3\}}_{P_3}(x,y).$$

The predicate $\text{query}^{\{3\}}_{P_3}(\cdot,\cdot)$ *is used to store the answer, which has a supra-index* $\{3\}$ *to indicate that the third argument in the answer to* P_3 *is missing.*

- *As a final example, consider the graph pattern*

$$P_4 = \underbrace{((?X, name, ?Y)\ \text{OPT}\ (?X, phone, ?Z))}_{P_4^1}\ \text{AND}\ \underbrace{(?Z, phone\_company, ?W)}_{P_4^2},$$

where the constant phone_company indicates that a phone number is associated with a phone company. In this case, we first consider a set of Datalog$^{\neg s}$ *rules that define the answer to the sub-pattern* P_4^1*, which is stored in the predicates* $\text{query}_{P_4^1}(\cdot,\cdot,\cdot)$ *and* $\text{query}^{\{3\}}_{P_4^1}(\cdot,\cdot)$*, and to the sub-pattern* P_4^2*, which is stored in* $\text{query}_{P_4^2}(\cdot,\cdot)$*. We have already seen how these rules look like, and thus we skip their definition. Having the above predicates in place, we now use two rules to define the answer to* P_4*. The first rule considers the case of the individuals with phone numbers:*

$$\text{query}_{P_4^1}(x,y,z), \text{query}_{P_4^2}(z,w) \rightarrow \text{query}_{P_4}(x,y,z,w).$$

Moreover, the second rule used to define the answers to P_4 *considers the case of the individuals without phone numbers, where a join is not needed:*

$$\text{query}^{\{3\}}_{P_4^1}(x,y), \text{query}_{P_4^2}(z,w) \rightarrow \text{query}_{P_4}(x,y,z,w) \qquad (5)$$

Although P_4 *is a valid SPARQL query, it can be difficult to interpret because if a person has no phone number, then she gets all the phone companies associated to her. The rules used to encode* P_4 *make this phenomenon clear: the two predicates in the body of rule (5) do not have any variables in common, so every pair of values assigned to variables* x, y *is combined with every pair of values assigned to* z, w*.*
$\square$

The approach in Example 1 can be generalized to represent any graph pattern P. We can construct a Datalog$^{\neg s}$ query $P_{\text{dat}} = (\Pi, \text{answer}_P)$, where Π is the union of

1. $\tau_{\text{bgp}}(P)$ that encodes the basic graph patterns occurring in P.

2. $\tau_{\text{opr}}(P)$ that represents the non-basic graph patterns occurring in P; in fact, these rules are used to encode the semantics of the SPARQL operators appearing in P.
3. $\tau_{\text{out}}(P)$ that computes the output predicate answer_P.

Example 1 gives a pretty good idea of how the programs $\tau_{\text{bgp}}(P)$ and $\tau_{\text{opr}}(P)$ are defined (their precise definitions can be found in [5]). For the definition of $\tau_{\text{out}}(P)$, there is one issue that needs to be resolved. Assume that P_3 is the graph pattern in Example 1. In this case, we expect $\text{query}_{P_3}(\cdot,\cdot,\cdot)$ to be the output predicate. However, the predicate $\text{query}^{\{3\}}_{P_3}(\cdot,\cdot)$ is also used to collect some answers; more specifically, $\text{query}^{\{3\}}_{P_3}(x,y)$ is used to collect the answers to the query where z is not assigned a value. To deal with this issue, the following rules are included in $\tau_{\text{opr}}(P_3)$:

$$\text{query}_{P_3}(x,y,z) \;\rightarrow\; \text{answer}_{P_3}(x,y,z) \qquad \text{query}^{\{3\}}_{P_3}(x,y) \;\rightarrow\; \text{answer}_{P_3}(x,y,\star),$$

where $\star$ is a special constant used to represent the fact that some positions in a tuple have not been assigned values. Thus, $\text{answer}_{P_3}(\cdot,\cdot,\cdot)$ is the only output predicate in this example (the precise definition of $\tau_{\text{out}}(P)$ can be found in [5]). The Datalog$^{\neg s}$ query that represents the graph pattern P is $P_{\text{dat}} = (\tau_{\text{bgp}}(P) \cup \tau_{\text{opr}}(P) \cup \tau_{\text{out}}(P), \text{answer}_P)$.

It remains to explain how we can compute the answer to a graph pattern P over an RDF graph G, denoted $[\![P]\!]_G$, which consists of mappings from the variables in P to the URIs in G.[7] To this end, we need to recall an auxiliary notion. Let $\bar{c} = (c_1,\ldots,c_n)$ be a tuple constants that belongs to the evaluation of P_{dat} over $\tau_{\text{db}}(G)$. By construction, in the set of rules $\tau_{\text{out}}(P)$ there exists an atom $\text{answer}_P(x_1,\ldots,x_n)$ that contains only variables (and not the constant $\star$). We define a mapping $\mu_{\bar{c},P}$ corresponding to $\bar{c}$ given P with its domain being the set of variables $\{x_i \mid i \in \{1,\ldots,n\}$ and $t_i \neq \star\}$ and, for every $i \in \{1,\ldots,n\}$, $t_i \neq \star$ implies $\mu_{\mathbf{t},P}(x_i) = c_i$. We then define the set of mappings corresponding to the answers of P_{dat} given $\tau_{\text{db}}(G)$: $[\![P_{\text{dat}}, \tau_{\text{db}}(G)]\!] = \{\mu_{\bar{c},P} \mid \bar{c} \in P_{\text{dat}}(\tau_{\text{db}}(G))\}$. It is not difficult to show, by induction on the structure of P, that:

Theorem 1.1 *For every graph pattern P and RDF graph G, $[\![P]\!]_G = [\![(P_{\text{dat}}, \tau_{\text{db}}(G))]\!]$.*

4.2. *SPARQL Entailment Regime and* TriQ-Lite 1.0

We now discuss how the above translation can be extended to encode using TriQ-Lite 1.0 the entailment regime to deal with RDFS and OWL vocabularies [17,20].

Storing Ontologies in RDF. We first recall the fragment of OWL 2 QL that includes the main features of the description logic DL-Lite$_{\mathcal{R}}$ [13], on which the profile OWL 2 QL is based. The vocabulary Σ of an OWL 2 QL core ontology is a finite set of unary and binary predicates, called classes and properties, respectively. A basic property over Σ is either p or p^-, where p is a property in Σ, while a basic class over Σ is either a or $\exists r$, where a is a class in Σ and r is a basic property over Σ. To represent an OWL 2 QL core ontology over a vocabulary Σ, we first include the following triples:

[7] Let us remind the reader that we refer to the definition of SPARQL 1.0 as defined in [26].

OWL 2 QL core Axiom	**RDF Triple**
SubClassOf(b_1, b_2)	$(b_1$, rdfs:subClassOf, $b_2)$
SubObjectPropertyOf(r_1, r_2)	$(r_1$, rdfs:subPropertyOf, $r_2)$
DisjointClasses(b_1, b_2)	$(b_1$, owl:disjointWith, $b_2)$
DisjointObjectProperties(r_1, r_2)	$(r_1$, owl:propertyDisjointWith, $r_2)$
ClassAssertion(b, a)	$(a$, rdf:type, $b)$
ObjectPropertyAssertion(p, a_1, a_2)	(a_1, p, a_2)

Table 1. Representation of OWL 2 QL core axioms as RDF triples.

- For every class a in Σ, we include the triple

  $(a$, rdf:type, owl:Class$)$.

  Notice that this triple uses the URIs rdf:type and owl:Class, and indicates that a, which is also a URI, is of type class.

- For every property p in Σ, we include the following triples, where p, p^-, $\exists p$ and $\exists p^-$ are considered as URIs (constants), and they are pairwise distinct:

  $(p$, rdf:type, owl:ObjectProperty$)$ $(p^-$, rdf:type, owl:ObjectProperty$)$

  indicating that p and p^- are properties,

  $(p$, owl:inverseOf, $p^-)$ $(p^-$, owl:inverseOf, $p)$

  indicating that p^- is the inverse of p and vice versa,

  $(\exists p$, rdf:type, owl:Restriction$)$ $(\exists p^-$, rdf:type, owl:Restriction$)$
  $(\exists p$, owl:onProperty, $p)$ $(\exists p^-$, owl:onProperty, $p^-)$
  $(\exists p$, owl:someValueFrom, owl:Thing$)$ $(\exists p^-$, owl:someValueFrom, owl:Thing$)$

  indicating that $\exists p$ and $\exists p^-$ are restrictions of p and p^-, respectively, and finally

  $(\exists p$, rdf:type, owl:Class$)$ $(\exists p^-$, rdf:type, owl:Class$)$

  indicating that $\exists p$ and $\exists p^-$ are classes.

We now discuss how OWL 2 QL core ontologies are stored as RDF graphs, following the standard syntax to represent OWL 2 ontologies as RDF triples [25]. By using the functional-style syntax of OWL [24], we can have the following axioms:

- SubClassOf(b_1, b_2): a basic class b_1 is a sub-class of a basic class b_2.
- SubObjectProperty(r_1, r_2): r_1 is a subproperty of r_2, where r_1, r_2 are basic properties.
- DisjointClasses(b_1, b_2): basic classes b_1 and b_2 are disjoint.
- DisjointObjectProperties(r_1, r_2): basic properties r_1 and r_2 are disjoint.
- ClassAssertion(b, a): a constant a belongs to a basic class b.

- ObjectPropertyAssertion(p, a_1, a_2): a constant a_1 is related to a constant a_2 via a property p.

Moreover, by following the mapping from [25], the above axioms are stored as RDF triples as in Table 1. An RDF graph G represents an OWL 2 QL core ontology if there is an OWL 2 QL core ontology $\mathcal{O}$ such that its representation as RDF generates G.

OWL 2 QL Core Direct Semantics Entailment Regime. We now discuss how a graph pattern is evaluated under the OWL 2 QL core direct semantics entailment regime, which is based on the definition of a direct semantics entailment regime for SPARQL 1.1 given in [17]. To compute the answer to a graph pattern, this regime is first applied at the level of basic graph patterns, and then the results of this step are combined using the standard semantics for the SPARQL operators [20]. Therefore, only the OWL 2 QL core direct semantics entailment regime for basic graph patterns needs to be defined.

Consider a basic graph pattern P. Under the OWL 2 QL core direct semantics entailment regime, the evaluation of P over an RDF graph G adopts an active domain semantics, that is, it uses the notion of entailment in OWL 2 QL core (which corresponds to the notion of entailment in DL-Lite$_{\mathcal{R}}$), but allowing the variables and blank nodes in P to take only values from G. For example, consider the RDF graph G:

$$(\text{dog}, \text{rdf:type}, \text{animal}) \qquad (\text{animal}, \text{rdfs:subClassOf}, \exists\text{eats}), \tag{6}$$

which indicate that dog is an animal, and every animal eats something. Moreover, assume that we want to retrieve the list of elements of G that eat something. The natural way to formulate this query is by using a graph pattern of the form $(?X, \text{eats}, \_ : B)$, where $\_ : B$ is a blank node. However, the answer to this query is empty under the OWL 2 direct semantics entailment regime, as there are no elements a, b in G that can be assigned to $?X$ and $\_ : B$ in such a way that the triple (a, eats, b) is implied by the axioms in G. In other words, the answer to $(?X, \text{eats}, \_ : B)$ is empty under the active domain semantics adopted in SPARQL 1.1. To obtain a correct answer in this case, we can consider the graph pattern $(?X, \text{rdf:type}, \exists\text{eats})$, as the triples in G can be used to infer the triple $(\text{dog}, \text{rdf:type}, \exists\text{eats})$, from which the correct answer dog is obtained.

Let G be an RDF graph representing an OWL 2 QL core ontology. Given a triple of URIs $\bar{u}$, we write $G \models \bar{u}$ to say that $\bar{u}$ is implied by G as defined in [17,23], which in turn is based on the notion of entailment for DL-Lite$_{\mathcal{R}}$ [13]. Moreover, given a basic graph pattern P, the evaluation of P over G under the OWL 2 QL core direct semantics entailment regime, denoted by $[\![P]\!]^{\mathbf{U}}_G$, is defined as the set of mappings μ from the variables in P to URIs from G such that, for every triple $\bar{u}$ of URIs that belongs to $\mu(P')$, with P' being a basic graph pattern obtained from P after replacing the blank nodes with URIs from G, $G \models \bar{u}$. Let us clarify that $\mathbf{U}$ in $[\![P]\!]^{\mathbf{U}}_G$ indicates that every blank node in P has to be assigned a URI from G. Now, the evaluation of an arbitrary graph pattern P over an RDF graph G under the OWL 2 QL core direct semantics entailment regime, denoted by $[\![P]\!]^{\mathbf{U}}_G$, is recursively defined as the usual semantics for graph patterns (see [26] for details), but replacing the rule for evaluating basic graph patterns by what has been described above.

There is a *fixed* Datalog$^{\exists,\neg s,\bot}$ program $\tau_{\text{owl2ql\_core}}$ that encodes the semantics $[\![\cdot]\!]^{\mathbf{U}}_G$. In this program, some simple Datalog rules are used to store in a unary predicate C all the URIs from the graph (we assume that an RDF graph does not contain blank nodes):

$$\text{triple}(x, y, z) \to \text{C}(x) \qquad \text{triple}(x, y, z) \to \text{C}(y) \qquad \text{triple}(x, y, z) \to \text{C}(z). \quad (7)$$

Then some Datalog rules are used to store the different elements in the ontology:

$$\begin{aligned}
\text{triple}(x, \text{rdf:type}, y) &\to \text{type}(x, y) \\
\text{triple}(x, \text{rdfs:subPropertyOf}, y) &\to \text{sp}(x, y) \\
\text{triple}(x, \text{owl:inverseOf}, y) &\to \text{inv}(x, y) \\
\text{triple}(x, \text{rdf:type}, \text{owl:Restriction}) & \\
\text{triple}(x, \text{owl:onProperty}, y) & \\
\text{triple}(x, \text{owl:someValueFrom}, \text{owl:Thing}) &\to \text{restriction}(x, y) \\
\text{triple}(x, \text{rdfs:subClassOf}, y) &\to \text{sc}(x, y) \\
\text{triple}(x, \text{owl:disjointWith}, y) &\to \text{disj}(x, y) \\
\text{triple}(x, \text{owl:propertyDisjointWith}, y) &\to \text{disj\_property}(x, y) \\
\text{triple}(x, y, z) &\to \text{triple}_1(x, y, z).
\end{aligned}$$

If we have the triples $(a, \text{rdf:type}, b)$ and $(b, \text{rdfs:subClassOf}, \exists r)$ in an OWL 2 QL core ontology, then the Datalog$^{\exists,\neg s,\bot}$ program $\tau_{\text{owl2ql\_core}}$ will create a triple of the form (a, r, ν), where ν is a null value. If (a, r, ν) is stored in the relation triple, then by the rules in (7) we get that $\text{C}(\nu)$ holds, violating the intended interpretation of the predicate C. To solve this problem, the Datalog rule $\text{triple}(x, y, z) \to \text{triple}_1(x, y, z)$ is used to produce a copy of $\text{triple}(\cdot, \cdot, \cdot)$ in the predicate $\text{triple}_1(\cdot, \cdot, \cdot)$. In this way, the new values are added to $\text{triple}_1(\cdot, \cdot, \cdot)$, that is, we do not modify the predicate $\text{triple}(\cdot, \cdot, \cdot)$ but instead both $\text{triple}_1(a, \text{rdf:type}, b)$ and $\text{triple}_1(b, \text{rdfs:subClassOf}, \exists r)$ hold, from which we conclude that $\text{triple}_1(a, r, \nu)$ also holds. Moreover, we have

$$\begin{aligned}
\text{sp}(x_1, x_2), \text{inv}(y_1, x_1), \text{inv}(y_2, x_2) &\to \text{sp}(y_1, y_2) \\
\text{type}(x, \text{owl:ObjectProperty}) &\to \text{sp}(x, x) \\
\text{sp}(x, y), \text{sp}(y, z) &\to \text{sp}(x, z)
\end{aligned}$$

to reason about properties. The first rule states that if p is a sub-property of q, then p^- is a sub-property of q^-. The other rules state that sub-property is reflexive and transitive. We also have the Datalog rules

$$\begin{aligned}
\text{sp}(x_1, x_2), \text{restriction}(y_1, x_1), \text{restriction}(y_2, x_2) &\to \text{sc}(y_1, y_2) \\
\text{type}(x, \text{owl:Class}) &\to \text{sc}(x, x) \\
\text{sc}(x, y), \text{sc}(y, z) &\to \text{sc}(x, z).
\end{aligned}$$

The first rule states that if p is a sub-property of q, then $\exists p$ is a sub-class of $\exists q$. The other rules state that sub-class is reflexive and transitive. We also have the rules

$$\begin{aligned}
\text{disj}(x_1, x_2), \text{sc}(y_1, x_1), \text{sc}(y_2, x_2) &\to \text{disj}(y_1, y_2) \\
\text{disj\_property}(x_1, x_2), \text{sp}(y_1, x_1), \text{sp}(y_2, x_2) &\to \text{disj\_property}(y_1, y_2).
\end{aligned}$$

to reason about disjointness. Finally, the following rules, which crucially use the features $\exists$ and $\bot$, are included to reason about membership assertions:

$$
\begin{aligned}
\text{triple}_1(x,u,y), \text{sp}(u,v) &\rightarrow \text{triple}_1(x,v,y)\\
\text{triple}_1(x,u,y), \text{inv}(u,v) &\rightarrow \text{triple}_1(y,v,x)\\
\text{type}(x,y), \text{restriction}(y,u) &\rightarrow \exists z\, \text{triple}_1(x,u,z)\\
\text{type}(x,y) &\rightarrow \text{triple}_1(x,\text{rdf:type},y)\\
\text{type}(x,y), \text{sc}(y,z) &\rightarrow \text{type}(x,z)\\
\text{triple}_1(x,u,y), \text{restriction}(z,u) &\rightarrow \text{type}(x,z)\\
\text{type}(x,y), \text{type}(x,z), \text{disj}(y,z) &\rightarrow \bot\\
\text{triple}_1(x,u,y), \text{triple}_1(x,v,y),&\\
\text{disj\_property}(u,v) &\rightarrow \bot.
\end{aligned}
$$

Given a graph pattern P and an RDF graph G, to compute $[\![P]\!]^{\mathbf{U}}_G$ we need to include $\tau_{\text{owl2ql\_core}}$ in the Datalog$^{\neg s}$ query P_{dat} defined in Section 4.1. More precisely, we need to add to the program of P_{dat} the program $\tau_{\text{owl2ql\_core}}$, but taking into consideration the active domain semantics in the entailment regime just defined. For example, assume that P is the basic graph pattern $(?X, \text{eats}, \_ : B)$ and G is the RDF graph in (6) storing information about animals. Then $\tau_{\text{bgp}}(P)$ is the following rule:

$$\text{triple}(x, \text{eats}, y) \rightarrow \text{query}_P(x). \tag{8}$$

In order to combine this rule with $\tau_{\text{owl2ql\_core}}$, we first need to consider the fact that all the triples inferred by using the axioms in G are stored in the predicate $\text{triple}_1(\cdot,\cdot,\cdot)$. Thus, we need to replace $\text{triple}(\cdot,\cdot,\cdot)$ by $\text{triple}_1(\cdot,\cdot,\cdot)$ in (8). We also need to enforce the constraint that every variable and blank node in P can only take a value from G, which is done by including the predicate C:

$$\text{triple}_1(x, \text{eats}, y), \text{C}(x), \text{C}(y) \rightarrow \text{query}_P(x). \tag{9}$$

Thus, given a graph pattern P, let $\tau^{\mathbf{U}}_{\text{bgp}}(P)$ be the set of rules obtained from $\tau_{\text{bgp}}(P)$ by first replacing triple by triple_1 in every rule of $\tau_{\text{bgp}}(P)$, and then adding $\text{C}(x)$ in the body of every resulting rule ρ if x occurs in ρ. Finally, we define

$$P^{\mathbf{U}}_{\text{dat}} = (\tau_{\text{owl2ql\_core}} \cup \tau^{\mathbf{U}}_{\text{bgp}}(P) \cup \tau_{\text{opr}}(P) \cup \tau_{\text{out}}(P), \text{answer}_P).$$

It is not difficult to show the following:

Theorem 1.2 *Consider a graph pattern P, and an RDF graph G that represents an OWL 2 QL core ontology. Then $P^{\mathbf{U}}_{\text{dat}}$ is a* TriQ-Lite 1.0 *query, and* $[\![P]\!]^{\mathbf{U}}_G = [\![(P^{\mathbf{U}}_{\text{dat}}, \tau_{\text{db}}(G))]\!]$.

Let us stress that the program $\tau_{\text{owl2ql\_core}}$, which encodes the semantics $[\![\cdot]\!]^{\mathbf{U}}_G$ for basic graph patterns, is fixed and does not depend on the given graph pattern P. This implies that, for a new graph pattern P', we only need to compute the programs $\tau^{\mathbf{U}}_{\text{bgp}}(P')$, $\tau_{\text{opr}}(P')$ and $\tau_{\text{out}}(P')$ without altering $\tau_{\text{owl2ql\_core}}$. This is quite beneficial since, whenever the user wants to pose a new query, (s)he can use $\tau_{\text{owl2ql\_core}}$ as a black box.

4.3. Removing the Active Domain Restriction

Consider the basic graph pattern:

$$Q = \{(?X, \text{eats}, \_\!: B), (\_\!: B, \text{rdf:type}, \text{plant\_material})\},$$

which asks for the animals that eat some plant material, and assume that G is an RDF graph. Under the active domain semantics, a is an answer to Q over G if we can replace the blank node $\_\!: B$ by a specific plant material b such that G implies $(?X, \text{eats}, b)$. But what happens if such a concrete witness cannot be found in G, and we can only infer that a is an answer to Q by using the axioms in the ontology? For example, this could happen if G stores information only about herbivores, so it includes the axiom $(\exists\text{eats}^-, \text{rdfs:subClassOf}, \text{plant\_material})$. In this case, Q has to be replaced by

$$\{(?X, \text{rdf:type}, \exists\text{eats}), (\exists\text{eats}^-, \text{rdfs:subClassOf}, \text{plant\_material})\}$$

in order to obtain the correct answers. And even worse, what happens if the query has to be distributed over several RDF graphs, which is a very common scenario in the Web. Then the user is forced to use a graph pattern of the form:

$$\{(?X, \text{eats}, \_\!: B), (\_\!: B, \text{rdf:type}, \text{plant\_material})\} \text{ UNION}$$
$$\{(?X, \text{rdf:type}, \exists\text{eats}), (\exists\text{eats}^-, \text{rdfs:subClassOf}, \text{plant\_material})\},$$

in which some inferences have to be encoded. All these issues can be solved if we do not force $\_\!: B$ to take values only in G, as this allows us to use the initial basic graph pattern Q. This gives rise to the semantics $[\![P]\!]_G^{\text{ALL}}$ that is defined exactly as $[\![P]\!]_G^{\mathbf{U}}$, but considering every basic graph pattern as a conjunctive query, and treating blank nodes as existential variables that are not forced to take only values in G (they can take values in the interpretations of G).

At this point, one may be tempted to think that the semantics $[\![\cdot]\!]^{\text{ALL}}$ can be directly defined by transforming every basic graph pattern into a conjunctive query, which has to be evaluated over a DL ontology. In fact, this approach works well with our initial query Q, which can be transformed into the conjunctive query

$$\exists y(\text{eats}(x, y) \land \text{plant\_material}(y)).$$

However, there are simple queries for which this approach does not work. For instance, consider the basic graph pattern $(?X, \text{rdfs:subClassOf}, \exists\text{eats})$. Given that $?X$ is used to store class names, this pattern cannot be transformed into a conjunctive query in order to define its semantics; instead, we need to replace $?X$ by every class name C, and then verify whether the inclusion $C \sqsubseteq \exists\text{eats}$ is implied by the DL ontology in order to define its semantics. It turned out that the more natural semantics $[\![\cdot]\!]^{\text{ALL}}$ can be easily defined by using Datalog$^{\exists,\neg s,\bot}$, without the need of differentiate between variables that are used to store individuals, classes or properties.

Given a basic graph pattern P, let $\tau_{\text{bgp}}^{\text{ALL}}(P)$ be the rule obtained from $\tau_{\text{bgp}}^{\mathbf{U}}(P)$ by removing every atom of the form $\mathsf{C}(x)$ such that x is a variable associated to a blank node occurring in P. For example, assume that P is the basic graph pattern $(?X, \text{eats}, \_\!: B)$. Then $\tau_{\text{bgp}}^{\mathbf{U}}(P)$ is the rule (9), and thus $\tau_{\text{bgp}}^{\text{ALL}}(P)$ is the rule:

$$\text{triple}_1(x, \text{eats}, y), \text{C}(x) \rightarrow \text{query}_P(x).$$

Moreover, given a graph pattern P, define $\tau_{\text{bgp}}^{\text{ALL}}(P)$ as the Datalog program consisting of the rules $\tau_{\text{bgp}}^{\text{ALL}}(P_i)$ for every basic graph pattern P_i occurring in P. Finally, we define

$$P_{\text{dat}}^{\text{ALL}} = (\tau_{\text{owl2ql\_core}} \cup \tau_{\text{bgp}}^{\text{ALL}}(P) \cup \tau_{\text{opr}}(P) \cup \tau_{\text{out}}(P), \text{answer}_P).$$

With this simple modification of $P_{\text{dat}}^{\mathbf{U}}$, we can formally define the semantics $[\![\cdot]\!]^{\text{ALL}}$, i.e., given a graph pattern P and an RDF graph G, $[\![P]\!]_G^{\text{ALL}}$ is defined as $[\![(P_{\text{dat}}^{\text{ALL}}, \tau_{\text{db}}(G))]\!]$. Note that $P_{\text{dat}}^{\text{ALL}}$ is a TriQ-Lite 1.0 query, for every graph pattern P. Thus, TriQ-Lite 1.0 is expressive enough to represent the OWL 2 core direct semantics entailment regime, even if the active domain restriction is not imposed.

5. Conclusions

A tractable Datalog-based query language has been discussed, called TriQ-Lite 1.0, which is expressive enough to encode every SPARQL query under the entailment regime for OWL 2 QL core. Moreover, this language allows us to formulate SPARQL queries in a simpler way, as it can easily encode a more natural notion of entailment.

An interesting question is whether TriQ-Lite 1.0 is powerful enough to deal with the other two lightweight profiles of OWL 2, namely OWL 2 EL and OWL 2 RL, and if not, how it can be extended in order to obtain a unique tractable Datalog-based language that can deal with all the three lightweight profiles of OWL 2 in a uniform way. Moreover, it would be interesting to investigate whether TriQ-Lite 1.0 is powerful enough for dealing also with the bag semantics of SPARQL. A good starting point for such an investigation is the recent work [10], which studies Datalog under bag semantics.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
[2] F. Alkhateeb, J. Baget, and J. Euzenat. Extending SPARQL with regular expression patterns (for querying RDF). *Journal of Web Semantics*, 7(2):57–73, 2009.
[3] R. Angles and C. Gutierrez. The expressive power of SPARQL. In A. P. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. W. Finin, and K. Thirunarayan, editors, *The Semantic Web - ISWC 2008, 7th International Semantic Web Conference, Karlsruhe, Germany, October 26-30, 2008. Proceedings*, volume 5318 of *Lecture Notes in Computer Science*, pages 114–129. Springer, 2008.
[4] M. Arenas, G. Gottlob, and A. Pieris. Expressive languages for querying the semantic web. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 14–26, 2014.
[5] M. Arenas, G. Gottlob, and A. Pieris. Expressive languages for querying the semantic web. *ACM Transactions on Database Systems*, 43(3):13:1–13:45, 2018.
[6] M. Arenas, C. Gutierrez, and J. Pérez. Foundations of RDF databases. In S. Tessaris, E. Franconi, T. Eiter, C. Gutiérrez, S. Handschuh, M. Rousset, and R. A. Schmidt, editors, *Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School 2009, Brixen-Bressanone, Italy, August 30 - September 4, 2009, Tutorial Lectures*, volume 5689 of *Lecture Notes in Computer Science*, pages 158–204. Springer, 2009.
[7] J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. On rules with existential variables: Walking the decidability line. *Artificial Intelligence*, 175(9-10):1620–1654, 2011.
[8] P. Barceló. Querying graph databases. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 175–188, 2013.

[9] C. Beeri and M. Y. Vardi. The implication problem for data dependencies. In *Proceedings of the 8th International Colloquium on Automata, Languages and Programming*, pages 73–85, 1981.

[10] L. E. Bertossi, G. Gottlob, and R. Pichler. Datalog: Bag semantics via set semantics. In P. Barceló and M. Calautti, editors, *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal*, volume 127 of *LIPIcs*, pages 16:1–16:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[11] A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *Journal of Artificial Intelligence Research*, 48:115–174, 2013.

[12] A. Calì, G. Gottlob, T. Lukasiewicz, B. Marnette, and A. Pieris. Datalog±: A family of logical knowledge representation and query languages for new applications. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 228–242. IEEE Computer Society, 2010.

[13] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *Journal of Automated Reasoning*, 39(3):385–429, 2007.

[14] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer, 1990.

[15] V. Fionda, C. Gutierrez, and G. Pirrò. Semantic navigation on the web of data: specification of routes, web fragments and actions. In A. Mille, F. L. Gandon, J. Misselis, M. Rabinovich, and S. Staab, editors, *Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012*, pages 281–290. ACM, 2012.

[16] T. Furche, B. Linse, F. Bry, D. Plexousakis, and G. Gottlob. RDF querying: Language constructs and evaluation methods compared. In P. Barahona, F. Bry, E. Franconi, N. Henze, and U. Sattler, editors, *Reasoning Web, Second International Summer School 2006, Lisbon, Portugal, September 4-8, 2006, Tutorial Lectures*, volume 4126 of *Lecture Notes in Computer Science*, pages 1–52. Springer, 2006.

[17] B. Glimm and C. Ogbuji. SPARQL 1.1 entailment regimes, 2013. W3C Recommendation 21 March 2013.

[18] G. Gottlob and A. Pieris. Beyond SPARQL under OWL 2 QL entailment regime: Rules to the rescue. In Q. Yang and M. J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 2999–3007. AAAI Press, 2015.

[19] S. Harris and A. Seaborne. SPARQL 1.1 query language, 2013. W3C Recommendation 21 March 2013.

[20] I. Kollia, B. Glimm, and I. Horrocks. SPARQL query answering over owl ontologies. In G. Antoniou, M. Grobelnik, E. P. B. Simperl, B. Parsia, D. Plexousakis, P. D. Leenheer, and J. Z. Pan, editors, *The Semantic Web: Research and Applications - 8th Extended Semantic Web Conference, ESWC 2011, Heraklion, Crete, Greece, May 29-June 2, 2011, Proceedings, Part I*, volume 6643 of *Lecture Notes in Computer Science*, pages 382–396. Springer, 2011.

[21] L. Libkin, J. L. Reutter, and D. Vrgoc. Trial for RDF: adapting graph query languages for RDF data. In R. Hull and W. Fan, editors, *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, pages 201–212. ACM, 2013.

[22] D. L. McGuinness and F. van Harmelen. OWL web ontology language overview, 2004. W3C Recommendation 10 February 2004.

[23] B. Motik, B. C. Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. OWL 2 web ontology language profiles (second edition), 2012. W3C Recommendation 11 December 2012.

[24] B. Motik, P. F. Patel-Schneider, and B. Parsia. OWL 2 web ontology language structural specification and functional-style syntax (second edition), 2012. W3C Recommendation 11 December 2012.

[25] P. F. Patel-Schneider and B. Motik. OWL 2 web ontology language mapping to rdf graphs (second edition), 2012. W3C Recommendation 11 December 2012.

[26] J. Pérez, M. Arenas, and C. Gutiérrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems*, 34(3):16:1–16:45, 2009.

[27] J. Pérez, M. Arenas, and C. Gutierrez. nsparql: A navigational language for RDF. *Journal of Web Semantics*, 8(4):255–270, 2010.

[28] A. Polleres. From SPARQL to rules (and back). In C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, and P. J. Shenoy, editors, *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 787–796. ACM, 2007.

[29] E. Prud'hommeaux and A. Seaborne. SPARQL query language for rdf, 2008. W3C Recommendation

15 January 2008.

[30] J. L. Reutter, A. Soto, and D. Vrgoc. Recursion in SPARQL. In M. Arenas, Ó. Corcho, E. Simperl, M. Strohmaier, M. d'Aquin, K. Srinivas, P. Groth, M. Dumontier, J. Heflin, K. Thirunarayan, and S. Staab, editors, *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I*, volume 9366 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2015.

[31] S. Rudolph and M. Krötzsch. Flag & check: data access with monadically defined queries. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 151–162, 2013.

[32] S. Schenk. A SPARQL semantics based on Datalog. In J. Hertzberg, M. Beetz, and R. Englert, editors, *KI 2007: Advances in Artificial Intelligence, 30th Annual German Conference on AI, KI 2007, Osnabrück, Germany, September 10-13, 2007, Proceedings*, volume 4667 of *Lecture Notes in Computer Science*, pages 160–174. Springer, 2007.

[33] W3C OWL Working Group. OWL 2 web ontology language document overview (second edition), 2012. W3C Recommendation 11 December 2012.