Beyond SPARQL under OWL 2 QL Entailment Regime: Rules to the Rescue

Georg Gottlob[1] **Andreas Pieris**[2]

[1]Department of Computer Science, University of Oxford, UK
[2]Institute of Information Systems, Vienna University of Technology, Austria

georg.gottlob@cs.ox.ac.uk, pieris@dbai.tuwien.ac.at

Abstract

SPARQL is the de facto language for querying RDF data, since its standardization in 2008. A new version, called SPARQL 1.1, was released in 2013, with the aim of enriching the 2008 language with reasoning capabilities to deal with RDFS and OWL vocabularies, and a mechanism to express navigation patterns through regular expressions. However, SPARQL 1.1 is not powerful enough for expressing some relevant navigation patterns, and it misses a general form of recursion. In this work, we focus on OWL 2 QL and we propose TriQ-Lite 1.0, a tractable rule-based formalism that supports the above functionalities, and thus it can be used for querying RDF data. Unlike existing composite approaches, our formalism has simple syntax and semantics in the same spirit as good old Datalog.

1 Introduction

RDF is the W3C recommendation data model to represent information about World Wide Web resources, while SPARQL is the standard language for querying RDF data, since its standardization in 2008. One of the distinctive features of Semantic Web data is the existence of vocabularies with predefined semantics: the *RDF Schema (RDFS)* and the *Ontology Web Language (OWL)*, which can be used to derive logical conclusions from RDF graphs; hence, an RDF query language equipped with reasoning capabilities to deal with these vocabularies is desirable. In addition, navigational capabilities are vital for data models with an explicit graph structure such as RDF [Alkhateeb *et al.*, 2009; Arenas *et al.*, 2009; Pérez *et al.*, 2010; Fionda *et al.*, 2012], while recursive definitions are a key feature for graph query languages [Barceló, 2013; Libkin *et al.*, 2013]. Having an RDF query language available that combines the above key functionalities is of paramount importance for the development of the Semantic Web. This has been recognized by the W3C, which led to the release of SPARQL 1.1 in 2013 [Harris and Seaborne, 2013; Glimm and Ogbuji, 2013], that is, an extended version of the 2008 language with reasoning capabilities to deal with RDFS and OWL vocabularies, and a mechanism to express navigation patterns through regular expressions. However, there are still useful queries that cannot be expressed in SPARQL 1.1, due to the lack of general recursion [Libkin *et al.*, 2013].

To the best of our knowledge, the only language that supports the above features, focussing on the profile OWL 2 QL of OWL 2, while its query evaluation problem is tractable in data complexity, is the recently introduced rule-based language TriQ-Lite, the lite version of the highly expressive triple query (TriQ) language [Arenas *et al.*, 2014]. This language is based on Datalog$^{\exists,\neg s,\bot}$, that is, Datalog extended with existential quantification in rule-heads, stratified negation, and negative constraints with the falsum ($\bot$) in rule-heads. Unfortunately, TriQ-Lite suffers from a serious drawback, which may revoke its advantage as an expressive RDF query language, namely it is not a *plain* language. A query language is called plain if it allows the user to write a query as a single program in a simple non-composite syntax. An example of a plain query language is Datalog, 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. In fact, TriQ-Lite is a composite language, where the user is forced to split the query in several modules $\Pi_1, \ldots, \Pi_n$ so that each Π_i can be expressed by the fragment of Datalog$^{\exists,\neg s,\bot}$ that is underlying TriQ-Lite, while each pair (Π_i, Π_{i+1}) is bridged via a set Q_i of conjunctive queries (more details are given in Section 4).

From the above discussion, we conclude that an RDF query language that fulfills certain desiderata, which in turn guarantee its applicability in real Semantic Web applications, is currently missing. These desiderata are the following:

1. **Plainness:** simple syntax and semantics, with the aim of simplifying the definition of queries;
2. **Reasoning Capabilities:** express every SPARQL query under the entailment regime for OWL 2 QL;
3. **Recursive Definitions:** general form of recursion must be supported, and ideally Datalog must be incorporated;
4. **Efficiency:** query evaluation must be data tractable, and feasible by the use of standard database technology.

At this point, we would like to expose an additional (conceptual) shortcoming of SPARQL 1.1, which must be taken

Desideratum	How it is Achieved
Plainness	Standard Datalog-like language
Reasoning capabilities	$\exists$-quantification and $\bot$ in rule-heads
Recursive definitions	Incorporate full Datalog$^{\neg s}$
Efficiency	Reduction to UCQ evaluation
Reasoning-Query decoupling	Expressive joins in rule-bodies, and $\exists$-quantification in rule-heads

Table 1: TriQ-Lite 1.0.

into account during the designing of an RDF query language. Under the OWL 2 direct semantics entailment regime, the evaluation of a basic graph pattern over an RDF graph adopts the so-called active domain semantics, i.e., it uses the notion of entailment in OWL 2 QL, but allowing variables and blank nodes to take only values from the RDF graph. As discussed in [Arenas *et al.*, 2014], this forces the user to encode part of the reasoning in the actual query, which undoubtedly leads to unnatural and complex queries (an example is given in Section 3). Notably, TriQ-Lite provides the definition of the more natural entailment regime, where the active domain semantics is dropped. This is certainly an additional desideratum:

5. **Reasoning-Query Decoupling:** the entailment regime without the active domain semantics must be definable, with the aim of decoupling the reasoning from the query.

In this work, we focus on TriQ-Lite, which is a language in evolution, and we investigate how it can be transformed into a plain language without sacrificing any of the other desiderata. The outcome of our study is TriQ-Lite 1.0, the new version of TriQ-Lite, which is based on Datalog$^{\exists,\neg sg,\bot}$; we use the superscript $\neg sg$ (instead of $\neg s$) since, for our purposes, it suffices to focus on negation that, apart from being stratified, is also grounded, i.e., it can be used with predicates that can only store constants. The proposed formalism is part of Datalog$^{\pm}$, that is, a family of logical KR languages [Calì *et al.*, 2012a]. Although several interesting Datalog$^{\pm}$ languages can be found in the literature (see, e.g., [Baget *et al.*, 2011; Thomazo *et al.*, 2012; Leone *et al.*, 2012; Calì *et al.*, 2012b; 2013]), none of them fulfills all desiderata. Even though Datalog$^{\pm}$ languages inherit the plainness of Datalog, either they are not expressive enough for satisfying desiderata 2, 3 and 5, or very expressive and thus intractable. Hence, our key challenge was to define a Datalog$^{\pm}$ language that achieves the right balance between expressivity and complexity.

We present a new syntactic paradigm, which is underlying TriQ-Lite 1.0, called *wardedness*, that can be informally described as follows: all the *dangerous* body-variables, i.e., variables that may be bound by the program to non-constant values, and at the same time are propagated to the rule-head, occur in exactly one body-atom, called *ward*, that can interact with the rest of the rule-body only via harmless join variables, i.e., variables that are bound by the program to database constants. Our technical results can be summarized as follows:

- We introduce in Section 4 TriQ-Lite 1.0, which is based on warded Datalog$^{\exists,\neg sg,\bot}$, and show that it fulfills all desiderata; the technical reasons are given in Table 1.
- For the reasoning-query decoupling, apart from expressive joins among body-variables that are bound to non-constant values (this exposes one of the main limitations of the language underlying TriQ-Lite), it is vital to allow for existentially quantified variables in rule-heads; this issue is discussed in Section 5.
- We provide in Section 6 a formal justification for the necessity of introducing a new Datalog-based formalism. We establish, via a model-theoretic argument, that existing tractable formalisms are not able to encode the OWL 2 direct semantics entailment regime, and at the same time ensure the reasoning-query decoupling.
- Finally, in Section 7, we substantiate the design choices made in the definition of our formalism. In fact, we show that very mild extensions lead to EXPTIME-hardness.

Let us say that warded Datalog$^{\exists,\neg sg,\bot}$, the formalism underlying TriQ-Lite 1.0, is well-suited as a general-purpose KR language, since it extends Datalog with features that allow us to express OWL 2 QL and OWL 2 RL ontologies.

2 Relational Databases and Datalog$^{\exists,\neg s,\bot}$

Consider the following pairwise disjoint (infinite countable) sets: a set $\mathbf{C}$ of *constants*, a set $\mathbf{N}$ of *(labeled) nulls*, and a set $\mathbf{V}$ of *variables*. A *term* t is a constant, null, or variable. An *atom* has the form $p(t_1,\ldots,t_n)$, where p is an n-ary predicate, and t_i's are terms. For an atom $\underline{a}$, $dom(\underline{a})$ and $var(\underline{a})$ is the set of its terms and variables, respectively; these extend to sets of atoms. An *instance* I is a (possibly infinite) set of atoms $p(\mathbf{t})$, where $\mathbf{t}$ is a tuple of constants and labeled nulls. A *database* D is a finite instance where only constants occur.

A *Datalog$^{\exists,\neg}$ rule* ρ is an expression of the following form: $\underline{a}_1,\ldots,\underline{a}_n,\neg\underline{b}_1,\ldots,\neg\underline{b}_m \rightarrow \exists Y_1\ldots\exists Y_k\ \underline{c}$, where (with $\mathbf{A}=\{\underline{a}_1,\ldots,\underline{a}_n\}$ and $\mathbf{B}=\{\underline{b}_1,\ldots,\underline{b}_m\}$): $n\geqslant 1$ and $m,k\geqslant 0$; $\underline{a}_i$'s and $\underline{b}_j$'s are atoms with terms from $(\mathbf{C}\cup\mathbf{V})$; $var(\mathbf{B})\subseteq var(\mathbf{A})$; $var(\mathbf{A})\cap\{Y_1,\ldots,Y_k\}=\varnothing$; and $\underline{c}$ contains terms from $(\mathbf{C}\cup\{Y_1,\ldots,Y_k\}\cup var(\mathbf{A}))$. The set $\mathbf{A}$ is denoted $B^+(\rho)$, while $\mathbf{B}$ is denoted $B^-(\rho)$. The *body* of ρ, denoted $B(\rho)$, is defined as $(\mathbf{A}\cup\mathbf{B})$. The atom $\underline{c}$ is the *head* of ρ, denoted $H(\rho)$. A *Datalog$^{\exists,\neg}$ program* Π is a finite set of Datalog$^{\exists,\neg}$ rules. A *stratification* of Π is defined in the same way as for Datalog$^{\neg}$. We say that Π is *stratified* if there exists a stratification of Π. A *constraint* ν is an assertion $\underline{a}_1,\ldots,\underline{a}_n\rightarrow\bot$, where $n\geqslant 1$ and $\underline{a}_i$'s are atoms with terms from $(\mathbf{C}\cup\mathbf{V})$. The *body* of ν, denoted $B(\nu)$, is the set $\{\underline{a}_1,\ldots,\underline{a}_n\}$. A *Datalog$^{\exists,\neg,\bot}$ program* Π is a finite set of Datalog$^{\exists,\neg}$ rules and constraints. We denote by $ex(\Pi)$ the set of Datalog$^{\exists,\neg}$ rules in Π; Π is *stratified* if $ex(\Pi)$ is stratified. A *stratified Datalog$^{\exists,\neg,\bot}$ query* Q is a pair (Π,p), where Π is a stratified Datalog$^{\exists,\neg,\bot}$ program, and p is a predicate not in the set of predicates occurring in Π, denoted $sch(\Pi)$. For brevity, we write Datalog$^{\exists,\neg s,\bot}$ for stratified Datalog$^{\exists,\neg,\bot}$.

The semantics $\Pi(D)$ of a program Π over a database D is defined in a standard way via the *chase procedure*; for details see [Calì *et al.*, 2012a]. Let $Q=(\Pi,p)$ be a Datalog$^{\exists,\neg s,\bot}$ query, where p is an n-ary predicate. If $\Pi(D)$ is defined ($\Pi(D)$ may be undefined due to the constraints), then the evaluation of Q over D, denoted $Q(D)$, is defined as the set of tuples $\{\mathbf{t}\in\mathbf{C}^n \mid p(\mathbf{t})\in\Pi(D)\}$. The *data complexity* of query evaluation is calculated by considering the query fixed.

3 SPARQL over OWL 2 QL into Datalog$^{\exists,\neg s,\bot}$

As recently shown in [Arenas *et al.*, 2014], SPARQL queries under OWL 2 direct semantics entailment regime over OWL 2 QL ontologies can be embedded into Datalog$^{\exists,\neg s,\bot}$. In particular, for a graph pattern P and an RDF graph G, it is possible to construct a Datalog$^{\exists,\neg s,\bot}$ query Q_P such that the answer to P over G can be easily decoded from $Q_P(D_G)$, where $D_G = \{\mathit{triple}(a,b,c) \mid (a,b,c) \in G\}$. For the sake of completeness, we briefly recall how the query Q_P is constructed. The reader who is familiar with this construction can directly go to Section 4.

3.1 SPARQL into Datalog$^{\neg s}$

For brevity, we first disregard the direct semantics entailment regime, and we illustrate, via some simple examples taken from [Arenas *et al.*, 2014], the main ingredients of the translation of a SPARQL query into a Datalog$^{\neg s}$ query.

Example 1 Let P_1 be the graph pattern $(?X, \text{name}, ?Y)$, where name is a constant, which 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. P_1 is represented via the rule

$$\mathit{triple}(X, \text{name}, Y) \;\rightarrow\; \mathit{query}_{P_1}(X, Y).$$

The predicate query_{P_1} is used to store the answer to the graph pattern P_1. Assume now that P_2 is the graph pattern $(?X, \text{name}, B)$, where B is a blank node, which asks for the list of elements in G that have a name (we are not interested in retrieving the blank nodes). P_2 is represented via the rule

$$\mathit{triple}(X, \text{name}, Y) \;\rightarrow\; \mathit{query}_{P_2}(X).$$

Notice that the variable Y represents the blank node B, and it does not appear in the head since we are not interested in retrieving names. Finally, consider the graph pattern P_3

$$\underbrace{(?X, \text{name}, ?Y)}_{Q_1} \quad \text{OPT} \quad \underbrace{(?X, \text{phone}, ?Z)}_{Q_2},$$

where phone is a property constant. For every constant a in an RDF graph, P_3 asks for the name and phone of a, if a's phone number is available in G, and otherwise it asks only for the name of a. Q_1 and Q_2 are represented via the rules

$$\begin{array}{rcl} \mathit{triple}(X, \text{name}, Y) & \rightarrow & \mathit{query}_{Q_1}(X, Y) \\ \mathit{triple}(X, \text{phone}, Z) & \rightarrow & \mathit{query}_{Q_2}(X, Z). \end{array}$$

Having the predicates query_{Q_1} and query_{Q_2} in place, we can now represent P_3 as follows. First, we define a set of rules for the cases where the phone number is available:

$$\begin{array}{rcl} \mathit{query}_{Q_1}(X, Y), \mathit{query}_{Q_2}(X, Z) & \rightarrow & \mathit{query}_{P_3}(X, Y, Z) \\ \mathit{query}_{Q_1}(X, Y), \mathit{query}_{Q_2}(X, Z) & \rightarrow & \mathit{compatible}_{P_3}(X). \end{array}$$

The predicate $\mathit{compatible}_{P_3}$ is used to store the individuals with phone numbers, which will be used in the rule that takes care of the individuals without phone numbers:

$$\mathit{query}_{Q_1}(X, Y), \neg \mathit{compatible}_{P_3}(X) \;\rightarrow\; \mathit{query}^{\{3\}}_{P_3}(X, Y).$$

The superscript $\{3\}$ in the binary predicate query_{P_3} indicates that the third argument in the answer to P_3 is missing. ∎

The approach shown above can be generalized to represent any graph pattern. Let $P = \{\mathbf{t}_1, \ldots, \mathbf{t}_n\}$ be a basic graph pattern such that $\mathbf{t}_i = (u_i, v_i, w_i)$, for every $i \in [1, n]$, with $\{?X_1, \ldots, ?X_k\}$ be the set of variables in P. Assume that γ is a substitution such that for every symbol u occurring in P, $\gamma(u) = u$ if $u \in \mathbf{C}$, $\gamma(u) = X$ if $u = ?X$, and $\gamma(u)$ is a fresh variable if u is a blank node. Then, ρ^{bgp}_P is defined as the rule:

$$\begin{array}{l} \mathit{triple}(\gamma(u_1), \gamma(v_1), \gamma(w_1)), \ldots, \\ \quad \mathit{triple}(\gamma(u_n), \gamma(v_n), \gamma(w_n)) \;\rightarrow\; \mathit{query}_P(X_1, \ldots, X_k). \end{array}$$

For a non-basic graph pattern P, Π^{bgp}_P is the Datalog program consisting of the rules ρ^{bgp}_Q for each basic graph pattern Q in P. Moreover, Π^{opr}_P is defined as the Datalog$^{\neg s}$ program that represents the non-basic graph patterns in P; in fact, Π^{opr}_P encodes the semantics of the SPARQL operators occurring in P as explained in Example 1. For our purposes, the formal definition of Π^{opr}_P is not important, and therefore we skip it. However, it is crucial to say that Π^{opr}_P is a Datalog$^{\neg s}$ program. Finally, the program Π^{out}_P, which defines the output predicate ans_P, is constructed as follows. By construction, some atoms of the form $\mathit{query}^J_P(X_1, \ldots, X_k)$, where J is a set of indexes, occur in the head of some rules of Π^{opr}_P (see atom $\mathit{query}^{\{3\}}_P(X, Y)$ in Example 1); if $J = \varnothing$ we simply write $\mathit{query}_P(X_1, \ldots, X_k)$. Let m_P be the arity of the predicate query_P, i.e., the query-predicate without superscript, occurring in Π^{opr}_P. For every atom $\mathit{query}^J_P(X_1, \ldots, X_k)$ in Π^{opr}_P, the following rule occurs in Π^{out}_P:

$$\mathit{query}^J_P(X_1, \ldots, X_k) \;\rightarrow\; \mathit{ans}_P(t_1, \ldots, t_{m_P}),$$

where, for each $i \in J$, t_i is the special constant $\star$, and after eliminating all the occurrences of $\star$ from $(t_1, \ldots, t_n)$ the tuple $(X_1, \ldots, X_k)$ is obtained. For instance, due to the atom $\mathit{query}^{\{3\}}_P(X, Y)$ occurring in $\Pi^{opr}_{P_3}$, where P_3 is the graph pattern given in Example 1, the rule

$$\mathit{query}^{\{3\}}_P(X, Y) \;\rightarrow\; \mathit{ans}_P(X, Y, \star)$$

occurs in $\Pi^{out}_{P_3}$. Eventually, the Datalog$^{\neg s}$ query Q_P is defined as (Π_P, ans_P), were $\Pi_P = (\Pi^{bgp}_P \cup \Pi^{opr}_P \cup \Pi^{out}_P)$.

Let us now explain how the answer to a graph pattern P over an RDF graph G, denoted $[\![P]\!]_G$, can be obtained from $Q_P(D_G)$. Consider an atom $\underline{a} = \mathit{ans}_P(t_1, \ldots, t_{m_P})$, where $(t_1, \ldots, t_{m_P}) \in Q_P(D_G)$. By construction, in Π^{out}_P there exists an atom $\mathit{ans}_P(X_1, \ldots, X_{m_P})$ that contains only variables (and not the constant $\star$). The mapping $\mu_{\underline{a},P}$ corresponding to $\underline{a}$ given P is defined as follows: for each $i \in [1, m_P]$, if $t_i \neq \star$, then $X_i \rightarrow t_i$ belongs to $\mu_{\underline{a},P}$, and nothing else occurs in $\mu_{\underline{a},P}$. Moreover,

$$[\![Q_P(D_G)]\!] \;=\; \{\mu_{\underline{a},P} \mid \underline{a} = \mathit{ans}(\mathbf{t}) \text{ and } \mathbf{t} \in Q_P(D_G)\}$$

are the mappings corresponding to the answers of Q_P over D_G. As shown in [Arenas *et al.*, 2014], $[\![P]\!]_G = [\![Q_P(D_G)]\!]$.

3.2 SPARQL Entailment Regime

We now show how the above translation can be extended in order to translate every SPARQL query under OWL 2 direct semantics entailment regime over OWL 2 QL ontologies into

a Datalog$^{\exists,\neg s,\bot}$ query. We first demonstrate how OWL 2 QL ontologies are stored as RDF graphs. Notice that in the specification of OWL 2, it is defined a standard syntax to represent OWL 2 ontologies as RDF triples. However, for brevity, we use the simplified syntax of [Arenas *et al.*, 2014], having in mind that the approach described below can be adapted to the standard syntax. The vocabulary of an OWL 2 QL ontology consists of unary and binary predicates, called classes and properties, respectively. A basic property is either p or p^-, where p is a property. A basic class is either c or $\exists p$, where c is a class and p a property. To represent an OWL 2 QL ontology, we first include some triples to indicate the classes and the properties in the ontology. For every class c we have the triple $(c, \text{rdf:type}, \text{owl:Class})$, asserting that c is of type class. For every property p we have the following triples:

$(p, \text{rdf:type}, \text{owl:Prop})$ $\quad$ $(p, \text{owl:inv}, p^-)$
$(p^-, \text{rdf:type}, \text{owl:Prop})$ $\quad$ $(p^-, \text{owl:inv}, p)$
$(\exists p, \text{owl:rest}, p)$ $\quad$ $(\exists p, \text{rdf:type}, \text{owl:Class})$
$(\exists p^-, \text{owl:rest}, p^-)$ $\quad$ $(\exists p^-, \text{rdf:type}, \text{owl:Class})$.

As said, the notation used above is a simplified version of the notation used in OWL 2. In particular, owl:Prop, owl:inv and owl:rest correspond to the keywords owl:ObjectProperty, owl:inverseOf and owl:Restriction, respectively. Note also that a triple such as $(\exists p, \text{owl:rest}, p)$ is represented by means of several triples in OWL 2.

The axioms in an OWL 2 QL ontology are represented as follows. $(c_1, \text{rdfs:sc}, c_2)$ indicates that the basic class c_1 is a subclass of the basic class c_2, while $(p_1, \text{rdfs:sp}, p_2)$ indicates that the basic property p_1 is a subproperty of the basic property p_2. To indicate that two basic classes c_1 and c_2 are disjoint we use $(c_1, \text{owl:disj}, c_2)$. As above, we employ a simplified notation; in fact, rdfs:sc, rdfs:sp and owl:disj correspond to the keywords rdfs:subClassOf, rdfs:subPropertyOf and owl:disjointWith, respectively. Note that OWL 2 QL allows for the use of qualified existential quantification in the right-hand side of inclusion assertions, which is not captured by the above encoding. However, such kind of axioms can be simulated in an OWL 2 QL ontology through the use of unqualified existential quantification, auxiliary roles, and inclusions between roles.

The membership assertions in an OWL 2 QL ontology are represented as follows. The triple $(a, \text{rdf:type}, c)$ indicates that the constant a belongs to the basic class c, while the triple (a_1, p, a_2) asserts that the constant a_1 is related to the constant a_2 via the property p. In the sequel, we say that an RDF graph G represents an OWL 2 QL ontology if there exists an OWL 2 QL ontology $\mathcal{O}$ such that, after the translation of $\mathcal{O}$ into RDF we obtain G.

Consider now a graph pattern P, and an RDF graph G that represents an OWL 2 QL ontology. In what follows, we present the construction of the Datalog$^{\exists,\neg s,\bot}$ query $Q_P^{\mathbf{C}}$ such that the answer to P over G under the OWL 2 direct semantics entailment regime as defined in [Glimm and Ogbuji, 2013], denoted $[\![P]\!]_G^{\mathbf{C}}$, coincides with $[\![Q_P^{\mathbf{C}}(D_G)]\!]$. Notice that the superscript $\mathbf{C}$ is used to indicate that the active domain semantics is adopted. First, we give the crucial program Π_{OWL2QL}, that is, the fixed Datalog$^{\exists,\neg s,\bot}$ program that encodes the OWL 2 direct semantics entailment regime over OWL 2 QL ontologies. This program contains three rules that store in a predicate $dom(\cdot)$ all the constant occurring in G:

$$\begin{aligned} triple(X,Y,Z) &\rightarrow dom(X) \\ triple(X,Y,Z) &\rightarrow dom(Y) \\ triple(X,Y,Z) &\rightarrow dom(Z). \end{aligned}$$

It also contains some Datalog rules that store the different elements in the ontology:

$$\begin{aligned} triple(X, \text{rdf:type}, Y) &\rightarrow type(X,Y) \\ triple(X, \text{rdfs:sp}, Y) &\rightarrow sp(X,Y) \\ triple(X, \text{owl:inv}, Y) &\rightarrow inv(X,Y) \\ triple(X, \text{owl:rest}, Y) &\rightarrow rest(X,Y) \\ triple(X, \text{rdfs:sc}, Y) &\rightarrow sc(X,Y) \\ triple(X, \text{owl:disj}, Y) &\rightarrow disj(X,Y) \\ triple(X,Y,Z) &\rightarrow triple_1(X,Y,Z). \end{aligned}$$

The purpose of the last rule is to guarantee that in the predicate $triple$ are stored only constants occurring in G and not nulls. As we shall see, Π_{OWL2QL} may generate a triple that contains a null z. Therefore, if such a triple is stored in the predicate $triple$, and not in the auxiliary predicate $triple_1$, we will conclude that $dom(z)$ holds, which violates the intended meaning of the dom predicate. Moreover, Π_{OWL2QL} contains the following rules to reason about properties:

$$\begin{aligned} sp(X_1,X_2), inv(Y_1,X_1), inv(Y_2,X_2) &\rightarrow sp(Y_1,Y_2) \\ type(X, \text{owl:Prop}) &\rightarrow sp(X,X) \\ sp(X,Y), sp(Y,Z) &\rightarrow sp(X,Z), \end{aligned}$$

the following rules to reason about classes:

$$\begin{aligned} sc(X_1,X_2), rest(Y_1,X_1), rest(Y_2,X_2) &\rightarrow sc(Y_1,Y_2) \\ type(X, \text{owl:Class}) &\rightarrow sc(X,X) \\ sc(X,Y), sc(Y,Z) &\rightarrow sc(X,Z), \end{aligned}$$

and the following rule for disjointness constraints:

$$disj(X_1,X_2), sc(Y_1,X_1), sc(Y_2,X_2) \rightarrow disj(Y_1,Y_2).$$

Finally, Π_{OWL2QL} contains the following rules to reason about membership assertions:

$$\begin{aligned} &triple_1(X,U,Y), sp(U,V) \rightarrow triple_1(X,V,Y) \\ &triple_1(X,U,Y), inv(U,V) \rightarrow triple_1(Y,V,X) \\ &type(X,Y), rest(Y,U) \rightarrow \exists Z\, triple_1(X,U,Z) \\ &type(X,Y) \rightarrow triple_1(X, \text{rdf:type}, Y) \\ &type(X,Y), sc(Y,Z) \rightarrow type(X,Z) \\ &triple_1(X,U,Y), rest(Z,U) \rightarrow type(X,Z) \\ &type(X,Y), type(X,Z), disj(Y,Z) \rightarrow \bot. \end{aligned}$$

Having the program Π_{OWL2QL} in place, the query $Q_P^{\mathbf{C}}$ is defined as $(\Pi_{OWL2QL} \cup \Pi_P^{\mathbf{C}}, ans_P)$, where $\Pi_P^{\mathbf{C}}$ is obtained from Π_P, as defined in Section 3.1, by applying the following modification: for each rule $\rho \in \Pi_P^{bgp}$ (recall that Π_P^{bgp} is the part of Π_P that encodes the basic graph patterns of P), add in the body of ρ the atom $dom(X)$, for each variable X occurring in ρ, and replace each occurrence of the predicate $triple$ by $triple_1$. The new dom-atoms enforce the constraint that every variable and blank node in P can only take a value from G. As shown in [Arenas *et al.*, 2014], $[\![P]\!]_G^{\mathbf{C}} = [\![Q_P^{\mathbf{C}}(D_G)]\!]$.

3.3 Dropping the Active Domain Restriction

As shown by the following example, the active domain restriction forces the user to encode part of the reasoning in the query, which leads to unnatural and complex queries.

Example 2 Consider the OWL 2 QL ontology $\mathcal{O}$ which states that Tom is a person and each person has father, and let G be the RDF graph that represents $\mathcal{O}$. Assume that we want to retrieve the elements of G that have a father. One may be tempted to claim that this query can be expressed via the graph pattern $P = (?X, \text{father}, B)$, where B is a blank node. However, $[\![P]\!]_G^{\mathbf{C}} = \varnothing$ since there are no elements a, b in G such that the triple (a, father, b) is implied by the ontology. To obtain the expected answer, we have to consider the graph pattern $(?X, \text{rdf:type}, \exists\text{father})$, which means that we are forced to implicitly encode the fact that the triple $(\text{Tom}, \text{rdf:type}, \exists\text{father})$ is inferred by the ontology. ∎

As already discussed in [Arenas *et al.*, 2014], the solution to the above problem is to relax the semantics $[\![P]\!]_G^{\mathbf{C}}$ in such a way that blank nodes are treated as existentially quantified variables that are not forced to take only values in the RDF graph. Indeed, after this relaxation, the answer to the the graph pattern $P = (?X, \text{father}, B)$ over the RDF graph G that represents the ontology $\mathcal{O}$ in Example 2 will be Tom since a triple $(\text{Tom}, \text{father}, z)$, where $z \in \mathbf{N}$, is inferred by $\mathcal{O}$. The more general semantics $[\![P]\!]_G^{\mathbf{C},\mathbf{N}}$ has been proposed in [Arenas *et al.*, 2014]. Given a graph pattern P and an RDF graph G that represents an OWL 2 QL ontology, $[\![P]\!]_G^{\mathbf{C},\mathbf{N}}$ is defined as $[\![Q_P^{\mathbf{C},\mathbf{N}}(D_G)]\!]$, where $Q_P^{\mathbf{C},\mathbf{N}} = (\Pi_{OWL2QL} \cup \Pi_P^{\mathbf{C},\mathbf{N}}, ans_P)$ with $\Pi_P^{\mathbf{C},\mathbf{N}}$ be the program obtained from $\Pi_P^{\mathbf{C}}$ as follows: assuming that Π_P^{bgp} is the part of $\Pi_P^{\mathbf{C}}$ that encodes the basic graph patterns, for each rule $\rho \in \Pi_P^{bgp}$, eliminate the atoms of the form $dom(X)$, where X is a variable that appears in the body but not in the head of ρ, which means that X represents a blank node. The superscript $\mathbf{C}, \mathbf{N}$ indicates that blank nodes may be assigned null values of $\mathbf{N}$.

In view of the fact that the problem of querying OWL 2 QL ontologies using conjunctive queries with inequalities, without restricting the inequalities to compare only database constants, is undecidable [Gutiérrez-Basulto *et al.*, 2013], and since SPARQL allows for inequalities, one may think that the more general semantics $[\![P]\!]_G^{\mathbf{C},\mathbf{N}}$ leads to undecidability. This is not true since the OWL 2 direct semantics entailment regime is first applied at the level of basic graph patterns, and the results of this step, which is a finite database that contains only constants, are combined using the standard SPARQL operators. Since in basic graph patterns we do not have inequalities, the above undecidability result cannot be applied.

4 From Modular to Standard Queries

Recall that the present work aims at proposing TriQ-Lite 1.0, the new version of TriQ-Lite, that enjoys all crucial desiderata analyzed in Section 1. Towards this direction, we are going to define a standard (non-composite) Datalog$^{\exists,\neg s,\bot}$-based formalism that is powerful enough to express the queries $Q_P^{\mathbf{C}}$ and $Q_P^{\mathbf{C},\mathbf{N}}$, incorporates Datalog$^{\neg s}$, and at the same time its query evaluation problem is tractable in data complexity.

Let us recall that TriQ-Lite, proposed in [Arenas *et al.*, 2014], although is tractable and powerful enough for expressing $Q_P^{\mathbf{C}}$ and $Q_P^{\mathbf{C},\mathbf{N}}$, it is a non-plain language, where the user can place different parts of the query in different modules, while its semantics is defined in a modular way. For example, the query $Q_P^{\mathbf{C}} = (\Pi_{OWL2QL} \cup \Pi_P^{\mathbf{C}}, ans_P)$ can be expressed (following the syntax of [Arenas *et al.*, 2014]) as

$$MQ_P^{\mathbf{C}} \;=\; [(\Pi_{OWL2QL}, \Pi_P^{bgp}), (\Pi_P^{opr}, \Pi_P^{out})],$$

where Π_{OWL2QL} is the fixed Datalog$^{\exists,\neg s,\bot}$ program that encodes the OWL 2 direct semantics entailment regime as discussed in Section 3.2, while $\{\Pi_P^{bgp}, \Pi_P^{opr}, \Pi_P^{out}\}$ forms a partition of $\Pi_P^{\mathbf{C}}$, where the first program consists of the rules that represent the basic graph patterns, the second one contains the rules that encode the SPARQL operators, and the last one consists of the rules that define the predicate ans_P. The semantics of such a query over a database D_G is defined in a modular way. First, consider the instance

$$I_P^{bgp} \;=\; \{p_\rho(\mathbf{t}) \mid \mathbf{t} \in q_\rho(\Pi_{OWL2QL}(D_G))\}_{\rho \in \Pi_P^{bgp}},$$

where, for each rule $\rho : \varphi(\mathbf{X}, \mathbf{Y}) \rightarrow p_\rho(\mathbf{X})$ of Π_P^{bgp}, q_ρ is the conjunctive query $\exists\mathbf{Y}\varphi(\mathbf{X}, \mathbf{Y})$. Then, the semantics of $MQ_P^{\mathbf{C}}$ over D_G is defined as

$$MQ_P^{\mathbf{C}}(D_G) \;=\; \{\mathbf{t} \mid \mathbf{t} \in q_\rho(\Pi_P^{opr}(I_P^{bgp}))\}_{\rho \in \Pi_P^{out}},$$

where, for each $\rho \in \Pi_P^{out}$, q_ρ is defined as above.

It is evident that TriQ-Lite significantly deviates from the standard way of defining Datalog-like query languages, and this makes the comprehensibility of its syntax and semantics a difficult task. Let us clarify that a single module of TriQ-Lite is not powerful enough for expressing $Q_P^{\mathbf{C}}$ and $Q_P^{\mathbf{C},\mathbf{N}}$. This is a strong indication that the modular nature of this langauge only helps to gain the missing expressive power for capturing SPARQL queries under the OWL 2 direct semantics entailment regime over OWL 2 QL ontologies, by treating the various parts of the same query in a different way.

The crucial question that comes up is how TriQ-Lite can be transformed into a standard Datalog-like query language, which fulfills all the aforementioned desiderata. The rest of this section is devoted to give an answer to this question by introducing TriQ-Lite 1.0.

4.1 A Standard Datalog$^{\exists,\neg s,\bot}$ Language

After a careful syntactic analysis of the query programs of $Q_P^{\mathbf{C}}$ and $Q_P^{\mathbf{C},\mathbf{N}}$, for an arbitrary graph pattern P, we identified an interesting property regarding the dangerous variables of each rule that was decisive for the definition of our language. From previous studies, we know that the rule-variables that must be tamed, in order to guarantee a good computational behavior, are the body-variables that can be associated with null values, and at the same time are propagated to the rule-head [Baget *et al.*, 2011; Calì *et al.*, 2013]; these are the variables that we dubbed dangerous. The query programs of $Q_P^{\mathbf{C}}$ and $Q_P^{\mathbf{C},\mathbf{N}}$ enjoy the following property: for each rule ρ, its dangerous variables are isolated in a single atom of $B^+(\rho)$, and they can interact with the rest of the rule-body only via

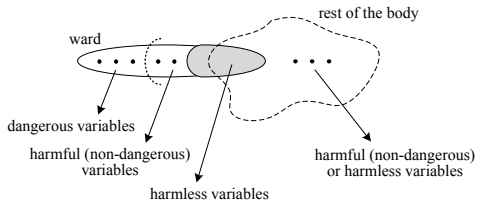


Figure 1: The anatomy of a warded rule-body.

harmless variables, i.e., variables that cannot be associated with null values. Another key observation is that the involved negation, apart from being stratified, is also grounded, i.e., it is used with predicates that can only store constant values, but not nulls. Let us now formalize the above key properties.

Technical Definitions. A *position* $p[i]$ identifies the i-th attribute of a predicate p. Given a Datalog$^{\exists,\neg s}$ program Π, the set of *affected positions* of $sch(\Pi)$, denoted $affected(\Pi)$, is inductively defined as follows: (1) if there exists $\rho \in \Pi$ such that at position π an existentially quantified variable occurs, then $\pi \in affected(\Pi)$; and (2) if there exists $\rho \in \Pi$, a variable X in $B^+(\rho)$ only at positions of $affected(\Pi)$, and X appears in $H(\rho)$ at position π, then $\pi \in affected(\Pi)$. Let $nonaffected(\Pi)$ be the nonaffected positions of $sch(\Pi)$. For a rule $\rho \in \Pi$, a variable $X \in var(B(\rho))$ is called *harmless* (w.r.t. Π) if at least one occurrence of X appears in $B^+(\rho)$ at a position of $nonaffected(\Pi)$; let $harmless(\rho)$ be the set of variables of $var(B(\rho))$ that are harmless. A variable $X \in var(B(\rho))$ is *dangerous* (w.r.t. Π) if $X \notin harmless(\rho)$ and $X \in var(H(\rho))$; let $dangerous(\rho)$ be the set of variables of $var(B(\rho))$ that are dangerous. Π is a Datalog$^{\exists,\neg sg}$ program ("sg" stands for stratified and ground) if, for each rule $\rho \in \Pi$, for each atom $p(t_1, \ldots, t_n) \in B^-(\rho)$, and for each $i \in [1, n]$, either $(t_i \in \mathbf{C})$ or $(t_i \in harmless(\rho))$.

Wardedness. We proceed to formalize the aforementioned property regarding the dangerous variables of a rule. Consider a Datalog$^{\exists,\neg sg}$ program Π and a rule $\rho \in \Pi$. A set of variables $\mathbf{X} \subseteq var(B(\rho))$ is called *warded* (w.r.t. Π) if, either $\mathbf{X} = \varnothing$, or there exists $\underline{a} \in B^+(\rho)$, called the *ward* (of $\mathbf{X}$ relative to ρ), such that: (1) $var(\underline{a}) \supseteq \mathbf{X}$; and (2) $(var(\underline{a}) \cap var(B(\rho) \setminus \{\underline{a}\})) \subseteq harmless(\rho)$. Intuitively, the ward isolates the harmful variables of $\mathbf{X}$ from the rest of the rule-body. Having the notion of the ward in place, it is now straightforward to formalize the intuitive idea of isolating the dangerous variables of a rule.

Definition 1 A Datalog$^{\exists,\neg sg,\bot}$ program Π is warded if, for each $\rho \in ex(\Pi)$, $dangerous(\rho)$ is warded. A Datalog$^{\exists,\neg sg,\bot}$ query (Π, p) is *warded* if Π is warded. A TriQ-Lite 1.0 query is a Datalog$^{\exists,\neg sg,\bot}$ query that is warded. ∎

The body of a rule occurring in a warded program can be graphically illustrated (via its hypergraph) as shown in Figure 1, where the shaded part consists only of harmless variables, while the dashed area represents an arbitrary (possibly cyclic) hypergraph. For every graph pattern P, both $Q_P^{\mathbf{C}}$ and $Q_P^{\mathbf{C},\mathbf{N}}$ are warded Datalog$^{\exists,\neg sg,\bot}$ queries, and thus:

Theorem 2 *Every SPARQL query under the OWL 2 direct semantics entailment regime over OWL 2 QL ontologies (with or without the active domain semantics) can be expressed as a* TriQ-Lite 1.0 *query.*

Warded Datalog$^{\exists,\neg sg,\bot}$, and therefore TriQ-Lite 1.0, can be conceived as a refined version of a more expressive formalism called weakly-frontier-guarded Datalog$^{\exists,\neg sg,\bot}$ [Baget *et al.*, 2011]. The latter requires the existence of a body-atom, called guard, that contains (or guards) all the dangerous variables. Thus, wardedness is a restriction of weak-frontier-guardedness, where the interface between the guard and the rest of the rule-body contains only harmless variables.

4.2 TriQ-Lite 1.0: A Tractable Language

Interestingly, the mild syntactic restriction applied on weakly-frontier-guarded Datalog$^{\exists,\neg sg,\bot}$ has a huge impact on the data complexity, namely it decreases from EXPTIME to PTIME. This implies that our formalism achieves the right balance between expressivity and complexity:

Theorem 3 *Query evaluation for* TriQ-Lite 1.0 *is* PTIME*-complete in data complexity.*

The lower bound immediately follows from the fact that plain Datalog is PTIME-hard [Dantsin *et al.*, 2001]. The nontrivial part is to establish the PTIME upper bound. Given a database D, and a TriQ-Lite 1.0 query $Q = (\Pi, p)$, we first need to check whether $\Pi(D)$ is defined. This task can be reduced to query evaluation for warded Datalog$^{\exists,\neg sg}$. Thus, it suffices to show that the latter is in PTIME. Fix a database D, and a warded Datalog$^{\exists,\neg sg}$ query $Q = (\Pi, p)$. Our goal is to construct in polynomial time a database D_L and a linear Datalog$^{\exists}$ query $Q_L = (\Pi_L, p)$ such that $Q(D) = Q_L(D_L)$, while the arity of the predicates of $sch(\Pi_L)$ does not depend on D. Linear Datalog$^{\exists}$ rules are rules with just one body-atom. Query evaluation for linear Datalog$^{\exists}$ is polynomial in the number of rules and the number of predicates occurring in the query program, and exponential in the maximum arity over all predicates in the query program [Gottlob *et al.*, 2014]. Thus, our reduction shows that query evaluation for warded Datalog$^{\exists,\neg sg}$ is in PTIME in data complexity. Moreover, query evaluation for linear Datalog$^{\exists}$ can be effectively reduced to query evaluation for unions of conjunctive queries [Gottlob *et al.*, 2014], and thus our reduction shows that query evaluation for warded Datalog$^{\exists,\neg sg}$ is feasible by the use of standard RDBMSs. Notice that a similar approach has been followed in [Arenas *et al.*, 2014] to show that query evaluation for TriQ-Lite is in PTIME in data complexity. However, as we shall discuss below, there is a fundamental difference between TriQ-Lite and warded Datalog$^{\exists,\neg sg}$ that makes the definition of the above reduction even more challenging.

The reduction to query evaluation for linear Datalog$^{\exists}$ consists of the following three steps: (1) **Normalization:** we normalize Π so that each rule is *head-ground*, i.e., each term occurring in the head is either a constant or a harmless variable, or *semi-body-ground*, i.e., there exists at most one body-atom that contains a harmful variable; (2) **Eliminate Negation:** we construct a database D^+, and a negation-free program Π^+ such that $Q(D) = Q^+(D^+)$, where $Q^+ = (\Pi^+, p)$. Since the negation is stratified and grounded, Π^+ can be computed from Π in the standard way, by replacing each negative atom

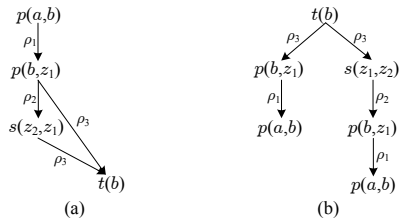


Figure 2: Proofs and proof-trees.

$\neg s(\mathbf{t})$ with $\bar{s}(\mathbf{t})$, where the extension of $\bar{s}$ in D^+ is the complement of s w.r.t. the ground semantics of Π over D, that is, the instance $\Pi(D)_\downarrow = \{\underline{a} \in \Pi(D) \mid dom(a) \subset \mathbf{C}\}$; and (3) **Linearization of Q^+:** We define $D_L = \Pi^+(D^+)_\downarrow$, and we transform Π^+ into Π_L. The program Π^+ can be partitioned into $\{\Pi_h^+, \Pi_b^+\}$, where Π_h^+ consists of all the head-ground rules of Π^+, while $\Pi_b^+ = (\Pi^+ \setminus \Pi_h^+)$ of semi-body-ground rules. The rules of Π_h^+ can be safely ignored since only ground atoms can be inferred from them, which are already in D_L. Finally, Π_L is obtained from Π_b^+ by replacing each non-linear rule ρ, where $\underline{a}$ is the ward of $dangerous(\rho)$, with the linear rules $\{h(\rho) \rightarrow h(head(\rho))_{h \in H_\rho}$, where H_ρ are all the homomorphisms that map $(body(\rho) \setminus \{\underline{a}\})$ to D_L.

Ground Semantics. It remains to show that the ground semantics can be computed in polynomial time. Fix a database D, and a warded Datalog$^\exists$ query $Q = (\Pi, p)$. We show that:

Proposition 4 *The instance $\Pi(D)_\downarrow$ can be constructed in polynomial time w.r.t. D.*

A similar result can be found in [Arenas *et al.*, 2014] for *constant-join* Datalog$^\exists$ programs, that is, programs consisting of rules that can join only harmless variables, which are underlying TriQ-Lite. However, warded Datalog$^\exists$ programs can join harmfull variables, and this is precisely the fundamental difference between constant-join Datalog$^\exists$ and warded Datalog$^\exists$ that makes our task technically more challenging. To establish Proposition 4, it suffices to show that the problem of deciding whether a ground atom $\underline{a}$ belongs to $\Pi(D)$ is feasible in polynomial time w.r.t. D. To this end, we significantly extend the alternating procedure Proof, proposed in [Arenas *et al.*, 2014], for constant-join Datalog$^\exists$. Let us first introduce the notion of the proof-tree via an example.

Example 3 Consider the warded Datalog$^\exists$ program Π':

$$\begin{array}{lcl} \rho_1 & : & p(X,Y) \rightarrow \exists Z\, p(Y,Z) \\ \rho_2 & : & p(X,Y) \rightarrow \exists Z\, s(Y,Z) \\ \rho_3 & : & p(X,Y), s(Y,Z) \rightarrow t(X), \end{array}$$

the database $D' = \{p(a,b)\}$, and the atom $\underline{a} = t(b)$. A proof of $\underline{a}$ w.r.t. D' and Π' is depicted in Figure 2(a), while a proof-tree of $\underline{a}$ w.r.t. D' and Π' is given in Figure 2(b). Having a proof of $\underline{a}$, we can construct a proof-tree of $\underline{a}$ by, roughly speaking, reversing the edges and unfolding the obtained graph into a tree by repeating some nodes. On the other hand, having a proof-tree of $\underline{a}$, we can construct a proof of $\underline{a}$ by reversing the edges and collapsing some nodes. ∎

Clearly, to establish Proposition 4, it suffices to show that the problem of deciding whether a proof-tree exists is feasible in polynomial time. Proof$(\underline{a}, D, \Pi)$ constructs a proof-tree P of $\underline{a}$ w.r.t. D and Π (if it exists) by constructing the various branches of P in parallel universal computations. The difficulty during this alternating procedure, is to guarantee that the branches constructed in parallel computations are compatible so that, after their merging, a valid proof-tree of $\underline{a}$ w.r.t. D and Π is obtained. In fact, we need to ensure that the "fresh" null values occurring in more than one branch of P (see the null z_1 in the proof-tree of Figure 2(b)), represent the same term in $\Pi(D)$. An abstract description of our algorithm follows.

Proof$(\underline{a}, D, \Pi)$ starts from $\underline{a}$, and applies appropriate resolution steps until the database D is reached. In particular, it consists of the following steps:

- A rule $\rho \in \Pi$ such that $H(\rho)$ and $\underline{a}$ unify is guessed. After resolving $\underline{a}$ with ρ we get the set of atoms $\theta_{\rho,\underline{a}}(B(\rho))$, where $\theta_{\rho,\underline{a}}$ is the most general unifier for $H(\rho)$ and $\underline{a}$. Notice that $\theta_{\rho,\underline{a}}$ maps the variables occurring in $B(\rho)$ but not in $H(\rho)$ into "fresh" nulls.
- $\theta_{\rho,\underline{a}}(B(\rho))$ is partitioned into $\{S_1, \ldots, S_n\}$ so that, for each null z occurring in $\theta_{\rho,\underline{a}}(B(\rho))$, there exists exactly one $i \in [1,n]$ such that S_i contains z, and there is no partition of $\theta_{\rho,\underline{a}}(B(\rho))$ with $n+1$ elements that satisfies the latter condition, i.e., each element of $\{S_1, \ldots, S_n\}$ is $\subseteq$-minimal. The intention underlying the above partition is to keep together the nulls that appear in more than one branch of the proof-tree under construction, until enough information regarding their generation, which will be used to ensure the compatibility among the various branches, is known.
- Universally select each set $S \in \{S_1, \ldots, S_n\}$ and prove it, which means that each atom of S must be proved. More precisely, for each atom $\underline{b} \in S$, a rule $\rho_{\underline{b}} \in \Pi$ is guessed such that $H(\rho_{\underline{b}})$ and $\underline{b}$ unify, and the set of atoms $\theta_{\rho_{\underline{b}},\underline{b}}(B(\rho_{\underline{b}}))$ is obtained.
- With $S = \{\underline{b}_1, \ldots, \underline{b}_m\}$, $\bigcup_{i \in [1,m]} \theta_{\rho_{\underline{b}_i},\underline{b}_i}(B(\rho_{\underline{b}_i}))$ is partitioned as above, and each component of the partition is proved recursively in a parallel universal computation.

During the execution of the above procedure, the first time that a null z is lost after resolving an atom $\underline{b}$ (that contains z) with a rule $\rho \in \Pi$, which means that z is associated with an existentially quantified variable in $H(\rho)$, we conclude that $\theta_{\rho,\underline{b}}(H(\rho))$ represents the atom where z is invented. It is vital to ensure that the atoms where z is invented in parallel computations are isomorphic to $\theta_{\rho,\underline{b}}(H(\rho))$. This can be achieved by carrying together with the component that contains z the crucial atom $\theta_{\rho,\underline{b}}(H(\rho))$. Although the above is only a high-level description of our algorithm, it gives enough evidence for its soundness. By exploiting wardedness, we can show that at each step of the computation of Proof$(\underline{a}, D, \Pi)$ we need logarithmic space w.r.t. D. Since ALOGSPACE coincides with PTIME, Proposition 4 follows.

5 Program Expressive Power

One may claim that Datalog$^{\neg s}$ and TriQ-Lite 1.0, or, equivalently, warded Datalog$^{\exists,\neg sg,\bot}$, are formalisms with the same expressive power, and raise the following question: is warded Datalog$^{\exists,\neg sg,\bot}$, and in particular the feature of existential quantification, necessary for our purposes? This question is

directly related to our fifth desideratum, i.e., the absolute decoupling between the reasoning process, and the process of defining the actual query. In fact, warded Datalog$^{\exists,\neg sg,\bot}$ fulfills the above desideratum since is able to encode the OWL 2 direct semantics entailment regime over OWL 2 QL ontologies via a fixed program, namely Π_{OWL2QL}, that does not depend on any graph pattern, and also is able to express every SPARQL query when the active domain semantics is dropped. However, the above desideratum, and in particular the encoding of the OWL 2 direct semantics entailment regime via a fixed program, cannot be achieved with Datalog$^{\neg s}$ [Arenas *et al.*, 2014]. Thus, the answer to the above question is that the existential quantification is vital in order to accomplish the decoupling between the reasoning part and the actual query.

The classical notion of expressive power does not capture the above key difference between Datalog$^{\neg s}$ and warded Datalog$^{\exists,\neg sg,\bot}$. Interestingly, this is captured by the recent notion of *program expressive power* [Arenas *et al.*, 2014]. Consider a query language $\mathcal{L}$ and a program Π. The program expressive power of Π relative to $\mathcal{L}$, denoted $\mathsf{Pep}_{\mathcal{L}}[\Pi]$, is the of triples $(D, \Lambda, p(\mathbf{t}))$ such that $(\Pi \cup \Lambda) \in \mathcal{L}$, the predicate p occurs only in the head of the rules of Λ (i.e., the rules of Λ act as the output rules of $(\Pi \cup \Lambda)$), and $\mathbf{t} \in Q(D)$, where $Q = (\Pi \cup \Lambda, p)$. Roughly, $\mathsf{Pep}_{\mathcal{L}}[\Pi]$ encodes that set of atoms that can be inferred from a database D via a query $Q \in \mathcal{L}$, where Π consists of the rules of Q other than the output rules. The program expressive power of $\mathcal{L}$ is now naturally defined as $\mathsf{Pep}[\mathcal{L}] = \{\mathsf{Pep}_{\mathcal{L}}[\Pi] \mid \Pi \in \mathcal{L}\}$. We say that a language $\mathcal{L}$ is more expressive than $\mathcal{L}'$ w.r.t. the program expressive power, written as $\mathcal{L}' \prec_{\mathsf{Pep}} \mathcal{L}$, if $\mathsf{Pep}[\mathcal{L}'] \subseteq \mathsf{Pep}[\mathcal{L}]$ and $\mathsf{Pep}[\mathcal{L}] \not\subseteq \mathsf{Pep}[\mathcal{L}']$. We can show that:

Theorem 5 *Datalog$^{\neg s}$* $\prec_{\mathsf{Pep}}$ TriQ-Lite 1.0.

The above result formally explains the key difference between Datalog$^{\neg s}$ and TriQ-Lite 1.0, and the importance of the existential quantification in rule-heads.

6 Model-theoretic Justification of Wardedness

The goal of this section is to justify the necessity for defining a new formalism instead of exploiting an existing one. To this end, we are going to establish, via a model-theoretic argument, that existing Datalog-based formalisms, which seem suitable for our purposes, are not able to encode the OWL 2 direct semantics entailment regime over OWL 2 QL ontologies via a *fixed* program. The only previously known plain formalisms that are powerful enough for our purposes are weakly-guarded Datalog$^{\exists,\neg sg,\bot}$ [Calì *et al.*, 2013] and weakly-frontier-guarded Datalog$^{\exists,\neg sg,\bot}$ [Baget *et al.*, 2011]. However, these two formalisms are EXPTIME-complete in data complexity, and thus inherently intractable. One may suggest to exploit existing tractable subclasses of the above formalisms. We show that such subclasses, and in particular (frontier-)guarded Datalog$^{\exists,\neg sg,\bot}$, are not powerful enough for encoding the OWL 2 direct semantics entailment regime (no matter whether the active domain semantics is adopted or not) via a fixed program. This justifies the necessity for introducing warded Datalog$^{\exists,\neg sg,\bot}$. We first expose, via an example, a crucial model-theoretic property that a language must fulfill in order to be able to encode the OWL 2 direct semantics entailment regime over OWL 2 QL ontologies via a fixed program.

Example 4 Consider the basic classes $c_0, \ldots, c_n$, and the basic property p, and let $\mathcal{O}$ be the OWL 2 QL ontology that encodes the following: the constant a belongs to c_0, c_0 is a subclass of $\exists p$, $\exists p^-$ is a subclass of c_1, and c_i is a subclass of c_{i+1}, for each $i \in [1, n-1]$. It is not difficult to verify that, after encoding $\mathcal{O}$ as an RDF graph G, we infer the triples $(z, \text{rdf:type}, c_i)$, for each $i \in [1, n]$, where z is a "fresh" null. Observe that the null value z is connected, via joint occurrences in triples, to all the constants in G that represent classes. In other words, the number of constants that are connected to z *depends* on the RDF graph. ∎

It can be seen that a Datalog$^{\exists,\neg s,\bot}$ language is suitable for our purposes only if it allows us to connect an invented null, using a fixed program, with an unbounded number of constants, or, in other words, if it enjoys the so-called unbounded ground-connection property. The *ground-connection* of a null z to an instance I, denoted $gc(z, I)$, is the set of constants $\{c \in \mathbf{C} \mid \text{there exists } \underline{a} \in I \text{ such that } \{c, z\} \subseteq dom(a)\}$, that is, all the constants that jointly appear with z in an atom of I. A family of databases is a sequence $D^1, D^2, \ldots$, where each D^n is a database such that $|dom(D^n)| = n$. The formal definition of the above key property follows:

Definition 6 A Datalog$^{\exists,\neg s,\bot}$ language $\mathbb{C}$ enjoys the *unbounded ground-connection property (UGCP)*, if there exists a family of databases $D^1, D^2, \ldots$, and a program Π in $\mathbb{C}$ such that, for each $n > 0$, $\Pi(D^n)$ is defined, and $|gc(z, \Pi(D^n))|$ depends on n, for at least one null z in $\Pi(D^n)$. ∎

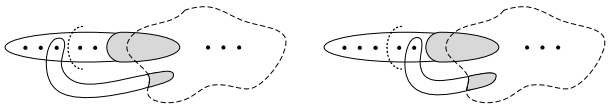
Our formalism enjoys the above model-theoretic property; this can be shown by exploiting Example 4. On the other hand, given a (frontier-)guarded Datalog$^{\exists,\neg sg,\bot}$ program Π, for every database D such that $\Pi(D)$ is defined, and for every null z in $\Pi(D)$, $gc(z, \Pi(D))$ consists of the constants that appear in the atom of $\Pi(D)$ in which z was generated, i.e., $|gc(z, \Pi(D))| \leqslant \omega$, where ω is the maximum arity over all predicates of $sch(\Pi)$, and thus does not depend on $|dom(D)|$. From the above discussion, we get the following:

Proposition 7 *It holds that, (1) Warded Datalog$^{\exists,\neg sg,\bot}$ enjoys the UGCP; and (2) (Frontier-)guarded Datalog$^{\exists,\neg sg,\bot}$ does not enjoy the UGCP.*

To sum up, all the known tractable subclasses of weakly-(frontier-)guarded Datalog$^{\exists,\neg sg,\bot}$ do not enjoy the UGCP, and thus they are not powerful enough for encoding the OWL 2 direct semantics entailment regime over OWL 2 QL ontologies via a fixed program. This justifies the necessity for defining warded Datalog$^{\exists,\neg sg,\bot}$, the Datalog-based language underlying TriQ-Lite 1.0.

7 Complexity-theoretic Justification of Wardedness

Let us now substantiate the design choices made in the definition of TriQ-Lite 1.0. We show that mild extensions of warded Datalog$^{\exists}$ immediately lead to languages that are EXPTIME-hard in data complexity, and thus provably intractable.



(a) Direct interaction ($\rightarrow$) (b) Indirect interaction ($\curvearrowright$)

Figure 3: A $\star$-warded rule-body, where $\star \in \{\rightarrow, \curvearrowright\}$.

This is a strong sign that wardedness lies at the boundaries of data tractability. Recall that the key idea underlying wardedness is to collect the dangerous variables of a rule in a single body-atom, and force them to interact with the rest of the rule-body only via harmless variables. Thus, the apparent way to extend warded Datalog$^\exists$ is to allow the dangerous variables to interact with the rest of the body via harmful ones.

Towards this direction, the mildest relaxation of wardedness is to allow the existence of an atom (other then the ward) in the body of a rule, which contains only harmless variables apart from one harmful variable V that is either dangerous or not. If V is dangerous, then we say that the interaction of the dangerous variables with the rest of the body is direct, and we get the formalism $\rightarrow$-warded Datalog$^\exists$; otherwise, we say that their interaction is indirect, and we get $\curvearrowright$-warded Datalog$^\exists$ ("$\rightarrow$" refers to the direct interaction, while "$\curvearrowright$" refers to the indirect interaction). The body of a rule occurring in a $\rightarrow$-warded (resp., $\curvearrowright$-warded) Datalog$^\exists$ program is graphically illustrated in Figure 3(a) (resp., 3(b)). Figures 1 and 3 confirm that the above proposed formalisms are obtained by minor syntactic extensions of warded Datalog$^\exists$. However, as we show below, these minor extensions have a significant impact on the data complexity of query evaluation.

Proposition 8 *Query evaluation for $\star$-warded Datalog$^\exists$, where $\star \in \{\rightarrow, \curvearrowright\}$, is* EXPTIME*-hard in data complexity.*

The above result is shown by simulating the behavior of an alternating polynomial space Turing machine M via a fixed $\star$-warded Datalog$^\exists$ query, where $\star \in \{\rightarrow, \curvearrowright\}$.

8 Future Work

Our next step is to experimentally evaluate the employed procedures. To this end, a challenging task is to design a practical algorithm for computing the ground semantics of a program. This will allows us to productively exploit the fact that query evaluation for TriQ-Lite 1.0 can be reduced to query evaluation for linear Datalog$^\exists$, which in turn implies that standard database technology and techniques can be employed.

Acknowledgements. G. Gottlob was supported by the EPSRC Programme Grant EP/M025268/ "VADA: Value Added Data Systems – Principles and Architecture". A. Pieris was supported by the Austrian Science Fund (FWF): P25207-N23 and Y698.

References

[Alkhateeb *et al.*, 2009] Faisal Alkhateeb, Jean-François Baget, and Jérôme Euzenat. Extending SPARQL with regular expression patterns (for querying RDF). *J. Web Sem.*, 7(2):57–73, 2009.

[Arenas *et al.*, 2009] Marcelo Arenas, Claudio Gutierrez, and Jorge Pérez. Foundations of RDF databases. In *RW*, pages 158–204, 2009.

[Arenas *et al.*, 2014] Marcelo Arenas, Georg Gottlob, and Andreas Pieris. Expressive languages for querying the semantic web. In *PODS*, pages 14–26, 2014.

[Baget *et al.*, 2011] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654, 2011.

[Barceló, 2013] Pablo Barceló. Querying graph databases. In *PODS*, pages 175–188, 2013.

[Calì *et al.*, 2012a] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012.

[Calì *et al.*, 2012b] Andrea Calì, Georg Gottlob, and Andreas Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012.

[Calì *et al.*, 2013] Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res.*, 48:115–174, 2013.

[Dantsin *et al.*, 2001] Evgeny Dantsin, Thomas Eiter, Gottlob Georg, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

[Fionda *et al.*, 2012] Valeria Fionda, Claudio Gutierrez, and Giuseppe Pirrò. Semantic navigation on the web of data: specification of routes, web fragments and actions. In *WWW*, pages 281–290, 2012.

[Glimm and Ogbuji, 2013] Birte Glimm and Chimezie Ogbuji. SPARQL 1.1 entailment regimes, 2013. W3C Recommendation 21 March 2013, http://www.w3.org/TR/sparql11-entailment/.

[Gottlob *et al.*, 2014] Georg Gottlob, Giorgio Orsi, and Andreas Pieris. Query rewriting and optimization for ontological databases. *ACM Trans. Database Syst.*, 39(3):25, 2014.

[Gutiérrez-Basulto *et al.*, 2013] Víctor Gutiérrez-Basulto, Yazmin Angélica Ibáñez-García, Roman Kontchakov, and Egor V. Kostylev. Conjunctive queries with negation over dl-lite: A closer look. In *RR*, pages 109–122, 2013.

[Harris and Seaborne, 2013] Steve Harris and Andy Seaborne. SPARQL 1.1 query language, 2013. W3C Recommendation 21 March 2013, http://www.w3.org/TR/sparql11-query/.

[Leone *et al.*, 2012] Nicola Leone, Marco Manna, Giorgio Terracina, and Pierfrancesco Veltri. Efficiently computable Datalog$^\exists$ programs. In *KR*, 2012.

[Libkin *et al.*, 2013] Leonid Libkin, Juan L. Reutter, and Domagoj Vrgoc. Trial for RDF: adapting graph query languages for RDF data. In *PODS*, pages 201–212, 2013.

[Pérez *et al.*, 2010] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. nSPARQL: a navigational language for RDF. *J. Web Sem.*, 8(4):255–270, 2010.

[Thomazo *et al.*, 2012] Michaël Thomazo, Jean-François Baget, Marie-Laure Mugnier, and Sebastian Rudolph. A generic querying algorithm for greedy sets of existential rules. In *KR*, 2012.