SPARQL-DL: SPARQL Query for OWL-DL

Evren Sirin[1] and Bijan Parsia[2]

[1] Clark & Parsia, LLC, Washington, DC
evren@clarkparsia.com
[2] Department of Computer Science, University of Manchester, UK
bparsia@cs.man.ac.uk

Abstract. There are many query languages (QLs) that can be used to query RDF and OWL ontologies but neither type is satisfactory for querying OWL-DL ontologies. RDF-based QLs (RDQL, SeRQL, SPARQL) are harder to give a semantics w.r.t. OWL-DL and are more powerful than what OWL-DL reasoners can provide. DL-based QLs (DIG ask queries, nRQL) have clear semantics but are not powerful enough in the general case. In this paper we describe SPARQL-DL, a substantial subset of SPARQL for which we provide a clear OWL-DL based semantics. SPARQL-DL is significantly more expressive than existing DL QLs (by allowing mixed TBox/RBox/ABox queries) and can still be implemented without too much effort on top of existing OWL-DL reasoners. We discuss design decisions and practical issues that arise for defining SPARQL-DL and report about our preliminary prototype implemented on top of OWL-DL reasoner Pellet.

1 Introduction

The query languages (QLs) for Semantic Web ontologies can be classified under two categories: RDF-based QLs and DL-based QLs. RDF-based query QLs, such as RDQL[3], SeRQL[4] and the upcoming W3C recommendation SPARQL [1], are based on the notion of RDF triple patterns and their semantics is based on matching triples with RDF graphs. It is harder to provide a semantics for these QLs under OWL-DL semantics because RDF representation mixes the syntax of the language with its assertions. The triple patterns in a query do not necessarily map to well-formed OWL-DL constructs. DL-based QLs such as the ASK queries of DIG protocol [2] or nRQL queries of RacerPro system [3], on the other hand, have well-defined semantics based on the DL model theory. However, DIG queries are limited to atomic (TBox or RBox or ABox) queries whereas nRQL supports only conjunctive ABox queries.

Our primary goal in this paper is to define a powerful and expressive query language for OWL-DL that can combine TBox/RBox/ABox queries. We also want our query language to align with SPARQL to improve the interoperability of applications on the Semantic Web. The other, rather competing, goal is to keep the query language simple enough that it can be easily built on top of existing OWL-DL reasoners.

To satisfy these requirements, we define SPARQL-DL, a substantial subset of SPARQL that can be covered by the standard reasoning services OWL-DL reasoners provide. We

[3] http://www.w3.org/Submission/RDQL/
[4] http://www.openrdf.org/doc/sesame/users/ch06.html

provide a semantics for SPARQL-DL which is directly based on the OWL-DL entailment relation. The semantics we present is also in agreement with the way the SPARQL specification envisions future extensions [1].

In the rest of the paper, we first present some background information for SPARQL and OWL-DL. We then describe several different ways SPARQL-DL can be defined and explain how we settled on our choice. We provide an abstract syntax for SPARQL-DL queries, define the semantics based on OWL-DL model theory, and provide a mapping from SPARQL-DL abstract syntax to RDF graph form, which can also be used for reverse transformation. There are various possibilities to extend our formulation of SPARQL-DL to allow more expressive and flexible queries. In Section 5, we analyze some of these possibilities and discuss the feasibility of such extensions. Throughout the paper we primarily focus on OWL-DL but SPARQL-DL can be directly used in conjunction with OWL-Lite or OWL 1.1 as we will discuss in Section 5.2.

2 Preliminaries

We will give a very brief overview of SPARQL and OWL-DL semantics in the next sections. Readers are referred to [1] and [4] for more detailed information. Throughout the paper we will use qnames to shorten URIs with rdf, rdfs, and owl prefixes to refer to standard RDF, RDF-S and OWL namespaces, respectively. We will also use the prefix ex to refer to an arbitrary example namespace.

2.1 SPARQL Syntax and Semantics

SPARQL is a query language developed primarily to query RDF graphs. The vocabulary for RDF graphs is three disjoint sets: a set of URIs $\mathcal{V}_{uri}$, a set of bnode identifiers $\mathcal{V}_{bnode}$, and a set of well-formed literals $\mathcal{V}_{lit}$. The union of these sets is called the set of RDF terms. An RDF triple is a tuple $(s, p, o) \in (\mathcal{V}_{uri} \cup \mathcal{V}_{bnode}) \times \mathcal{V}_{uri} \times (\mathcal{V}_{uri} \cup \mathcal{V}_{bnode} \cup \mathcal{V}_{lit})$. An RDF graph is a finite set of RDF triples.

The building block for SPARQL queries is *Basic Graph Patterns* (BGP). A SPARQL BGP is a set of *triple patterns*. A triple pattern is an RDF triple in which zero or more variables might appear. Variables are taken from the infinite set $\mathcal{V}_{var}$ which is disjoint from the above mentioned sets. A solution to a SPARQL BGP w.r.t. to a source RDF graph G is a mapping μ from the variables in the query to RDF terms such that the substitution of variables in the BGP would yield a subgraph of G (according to the definition of subgraph matching in RDF semantics [5]). Note that, the bnodes in the query (as bnodes in the source graph) are treated as existential variables, i.e. they are non-distinguished variables.

More complex SPARQL queries are constructed from BGPs by using projection (SELECT operator), left join (OPTIONAL operator), union (UNION operator) and constraints (FILTER operator). The semantics for these operations are defined as algebraic operations over the solutions of BGPs [6]. Thus, an alternative semantics that replaces simple graph matching criteria can be given by simply providing a different solution condition for BGPs as we will provide later in the paper.

2.2 OWL-DL Syntax and Semantics

OWL-DL, despite being based on RDF, has a quite different semantics. An OWL-DL vocabulary partitions the set $\mathcal{V}_{uri}$ into many mutually disjoint sets. Formally, an OWL-DL vocabulary $\mathcal{V}_{\mathcal{O}} = (\mathcal{V}_{cls}, \mathcal{V}_{op}, \mathcal{V}_{dp}, \mathcal{V}_{ap}, \mathcal{V}_{ind}, \mathcal{V}_{D}, \mathcal{V}_{lit})$ is a 7-tuple where $\mathcal{V}_{cls}$ is the set of URIs denoting class names, $\mathcal{V}_{op}$ is the set of URIs denoting object properties, $\mathcal{V}_{dp}$ is the set of URIs denoting datatype properties, $\mathcal{V}_{ap}$ is the set of URIs denoting annotation properties, $\mathcal{V}_{ind}$ is the set of URIs denoting individuals, $\mathcal{V}_{D}$ is the set of URIs denoting datatype names, and $\mathcal{V}_{lit}$ is the set of well-formed RDF literals. In OWL-DL, $\mathcal{V}_{uri}$ is the union of $\mathcal{V}_{cls}$, $\mathcal{V}_{op}$, $\mathcal{V}_{dp}$, $\mathcal{V}_{ap}$, $\mathcal{V}_{ind}$, and $\mathcal{V}_{D}$and does not include any of builtin URIs from RDF, RDF-S, or OWL namespace.

OWL-DL provides several constructs to generate complex class expressions from named classes. The set of OWL-DL classes (written S_c) is defined inductively using the following grammar:

$$C \leftarrow A \mid \mathit{not}(C) \mid \mathit{and}(C_1, \ldots, C_2) \mid \mathit{or}(C_1, \ldots C_2) \mid \{a\} \mid \mathit{some}(p, C) \mid \mathit{all}(p, C) \mid \\ \mathit{min}(n, q) \mid \mathit{max}(n, q) \mid \mathit{some}(t, D) \mid \mathit{all}(t, D) \mid \mathit{min}(n, t) \mid \mathit{max}(n, t)$$

where $A \in \mathcal{V}_{cls}$, $C_{(i)} \in S_c$, $a \in \mathcal{V}_{ind}$, $p \in \mathcal{V}_{op}$, $q \in \mathcal{V}_{op}$ and it is *simple*[5], $t \in \mathcal{V}_{dp}$, $D \in \mathcal{V}_{D}$, and n is a non-negative integer.

Let $\mathcal{V}_{\mathcal{O}}$ be an OWL vocabulary. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a tuple where $\Delta^{\mathcal{I}}$, the domain of discourse, is a union of two disjoint sets $\Delta^{\mathcal{I}}_{O}$ (the object domain) and $\Delta^{\mathcal{I}}_{D}$ (the data domain); and $\mathcal{I}$ is the interpretation function that gives meaning to the entities defined in the ontology. $\mathcal{I}$ maps each OWL class $C \in \mathcal{V}_{cls}$ to a subset $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}_{O}$, each object property $p \in \mathcal{V}_{op}$ to a binary relation $p^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}_{O} \times \Delta^{\mathcal{I}}_{O}$, each datatype property $t \in \mathcal{V}_{dp}$ to a binary relation $t^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}_{O} \times \Delta^{\mathcal{I}}_{D}$, each annotation property $p_a \in \mathcal{V}_{ap}$ to a binary relation $t^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, each individual $a \in \mathcal{V}_{ind}$ to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}_{O}$, each datatype $D \in \mathcal{V}_{D}$ to a subset $D^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}_{D}$, and each literal $l \in \mathcal{V}_{lit}$ to an element $l^{\mathcal{I}} \in \Delta^{\mathcal{I}}_{D}$. The interpretation function is extended to complex class expressions in the usual way as explained in detail in [4].

An ontology contains a finite number of class, property and individual axioms. There are many syntactic forms for axioms in OWL-DL, but most of these axioms are syntactic sugar and can be expressed as a combination of SubClassOf(C_1, C_2), SubPropertyOf(p_1, p_2) and Transitive(p) axioms. The following table shows the conditions under which an interpretation $\mathcal{I}$ satisfies an axiom[6]:

Axiom	**Condition on interpretation**
SubClassOf(C_1, C_2)	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
SubPropertyOf(p_1, p_2)	$p_1^{\mathcal{I}} \subseteq p_2^{\mathcal{I}}$
Transitive(p)	$\langle x, y\rangle \in p^{\mathcal{I}}$ and $\langle y, z\rangle \in p^{\mathcal{I}}$ implies $\langle x, z\rangle \in p^{\mathcal{I}}$

We say that an an ontology $\mathcal{O}$ is consistent if there exists an interpretation $\mathcal{I}$ that satisfies all the axioms in $\mathcal{O}$; such $\mathcal{I}$ is then called a model of $\mathcal{O}$. $\mathcal{O}$ entails an axiom α if every model of $\mathcal{O}$ satisfies α. $\mathcal{O}$ entails another ontology $\mathcal{O}'$ if every model of $\mathcal{O}$ is also a model of $\mathcal{O}'$.

[5] See [4] for a precise definition of simple roles

[6] Note that, in OWL-DL annotations are treated as special kind of axioms which are satisfied by an interpretation only if they explicitly exist in the ontology

3 Roadmap for SPARQL-DL

SPARQL is designed so that its graph matching semantics can be extended to an arbitrary entailment regime [1]. A graph in the range of an entailment regime E is called well-formed for the E-entailment. The condition for E-entailment is defined as follows in the SPARQL specification: After the variables in a BGP are substituted with constant terms, the BGP should be well-formed for E-entailment and entailed by the source document.

As explained in the previous section, there is a well-defined entailment relation between valid OWL-DL ontologies. Therefore, as a minimal condition, we would require the variable substitution to yield a valid OWL-DL ontology (without the OWL-DL requirement that every entity is typed because the source ontology is already supposed to have the typing information). However, if we allow the variables to be mapped to arbitrary URIs, especially to the ones from the builtin OWL vocabulary, we might end up having rather unusual queries although the substitution yields a valid OWL-DL ontology. For example, consider the query pattern

```
?C rdfs:subClassOf _:x .
_:x rdf:type owl:Restriction .
_:x owl:onProperty ex:q .
_:x ?p ?C .
```

where substituting the variable $?C$ with a URI denoting a class name and the variable $?p$ with owl:allValuesFrom or owl:someValuesFrom yields a valid OWL-DL ontology. This is a rather unusual query because the variable $?p$ ranges over the quantifiers of the language. Although such a query is not expressible in First Order Logic (FOL), answering this query is possible (assuming that the variable $?C$ can only be mapped to named classes). Since we can enumerate both the classes defined in the ontology and the quantifiers of the language, we can simply try every substitution and check for the entailment of the resulting axiom. However, such a query evaluation strategy would be highly inefficient and the utility of such queries is arguable.

There are other cases where we would like to ask queries that are not expressible in FOL. Consider a very simple query like:

```
ex:a rdf:type ?C .
```

that asks for the types of an individual ex:a. This is also a higher order query because a variable is used in the predicate position. Although this query is not expressible as a DL ABox query, a DL reasoner that provides the *realization* service can easily answer this query.

Our goal is to define an entailment regime for SPARQL based on OWL-DL which can be implemented in a rather straightforward way. For this reason, we adopt the strategy of how OWL-DL is defined: We first start by defining an abstract syntax for SPARQL-DL queries, describe its semantics, and then provide a transformation from the abstract syntax to the triple patterns of SPARQL.

4 SPARQL-DL

4.1 SPARQL-DL Abstract Syntax

Let $\mathcal{V}_{\mathcal{O}} = (\mathcal{V}_{cls}, \mathcal{V}_{op}, \mathcal{V}_{dp}, \mathcal{V}_{ap}, \mathcal{V}_{ind}, \mathcal{V}_D, \mathcal{V}_{lit})$ be an OWL-DL vocabulary. Let $\mathcal{V}_{bnode}$ and $\mathcal{V}_{var}$ be the set of bnode identifiers and set of variables as before. A SPARQL-DL query atom q is of the form:

$$\begin{aligned} q \leftarrow\ & \mathsf{Type}(a, C) \mid \mathsf{PropertyValue}(a, p, v) \mid \mathsf{SameAs}(a, b) \mid \mathsf{DifferentFrom}(a, b) \mid \\ & \mathsf{EquivalentClass}(C_1, C_2) \mid \mathsf{SubClassOf}(C_1, C_2) \mid \mathsf{DisjointWith}(C_1, C_2) \mid \\ & \mathsf{ComplementOf}(C_1, C_2) \mid \mathsf{EquivalentProperty}(p_1, p_2) \mid \mathsf{SubPropertyOf}(p_1, p_2) \mid \\ & \mathsf{InverseOf}(p_1, p_2) \mid \mathsf{ObjectProperty}(p) \mid \mathsf{DatatypeProperty}(p) \mid \mathsf{Functional}(p) \mid \\ & \mathsf{InverseFunctional}(p) \mid \mathsf{Transitive}(p) \mid \mathsf{Symmetric}(p) \mid \mathsf{Annotation}(s, p_a, o) \end{aligned}$$

where $a, b \in \mathcal{V}_{uri} \cup \mathcal{V}_{bnode} \cup \mathcal{V}_{var}$, $v \in \mathcal{V}_{uri} \cup \mathcal{V}_{lit} \cup \mathcal{V}_{bnode} \cup \mathcal{V}_{var}$, $p, q \in \mathcal{V}_{uri} \cup \mathcal{V}_{var}$, $C, D \in S_c \cup \mathcal{V}_{var}$, $s \in \mathcal{V}_{uri}$, $p_a \in \mathcal{V}_{ap}$, $o \in \mathcal{V}_{uri} \cup \mathcal{V}_{lit}$. A SPARQL-DL query Q is a finite set of SPARQL-DL query atoms and the query is interpreted as the conjunction of the elements in the set.

We can represent various different queries with the atoms we defined. Table 1 shows some example queries asked against a fictional ontology of universities. Query $Q1$ and $Q2$ are just examples of standard DL queries. Query $Q3$ shows how bnodes are used as non-distinguished variables. For example, suppose we have the following KB:

PropertyValue(person1,hasPublication,paper1),Type(paper1,ConferencePaper),
SubClassOf(ConferencePaper,some(publishedAt,Conference)),
DisjointWith(Conference,Workshop)

Then, query $Q3$ would have a solution even though we do not know at which conference the paper is published. Query $Q4$ shows a novel aspect of OWL-DL where we mix an ABox and a TBox query. In this query, we are not only asking for students that are also employees (as we did in $Q2$) but we also would like to learn what kind of employee they are (ResearchAssistant and AdministrativeAssistant being some of the possibilities). Query $Q5$ shows another possibility where we mix an RBox query with an ABox query to retrieve all the object property values for a specific individuals. Note that many different query types can be constructed by mixing different types of query atoms.

Query	Type of the query	An example query
$Q1$	Standard TBox query	SubClassOf($?c$, ex:Student)
$Q2$	Standard ABox query	Type($?x$, and(ex:Student, ex:Employee)), PropertyValue($?x$, ex:name, $?y$)
$Q3$	ABox query with non-distinguished variables	PropertyValue($?x$, ex:hasPublication, _:y), PropertyValue(_:y, ex:publishedAt, _:z), Type(_:z, not(ex:Workshop))
$Q4$	Mixed ABox/TBox query	Type($?x$, ex:Student), Type($?x$, $?c$) SubClassOf($?c$, ex:Employee),
$Q5$	Mixed ABox/RBox query	ObjectProperty($?p$), PropertyValue(ex:John, $?p$, $?v$)

Table 1. Example SPARQL-DL queries showing some possible uses

4.2 SPARQL-DL Semantics

The semantics of SPARQL-DL is very similar to the semantics of OWL-DL. We specify the conditions under which an interpretation satisfies a query atom in much the same way that satisfaction is defined for OWL-DL axioms. We will define the satisfaction for only atoms that have no distinguished variables (i.e. there might still be bnodes in the atom). We will call such atoms *semi-ground* query atoms.

Let $\mathcal{O}$ be an OWL-DL ontology, $\mathcal{V}_{\mathcal{O}} = (\mathcal{V}_{cls}, \mathcal{V}_{op}, \mathcal{V}_{dp}, \mathcal{V}_{ap}, \mathcal{V}_{ind}, \mathcal{V}_{D}, \mathcal{V}_{lit})$ the vocabulary for $\mathcal{O}$, and $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation for $\mathcal{O}$. We say that a semi-ground query atom is *compatible* with the vocabulary $\mathcal{V}_{\mathcal{O}}$ if all the URIs used in query atoms are typed correctly, e.g. for $\mathsf{Type}(a, C)$ we have $a \in \mathcal{V}_{ind}$ and $C \in S_c$ and so on.

We define an evaluation $\sigma : \mathcal{V}_{ind} \cup \mathcal{V}_{bnode} \cup \mathcal{V}_{lit} \to \Delta^{\mathcal{I}}$ to be a mapping from the individual names, bnodes, and literals used in the query to the elements of interpretation domain $\Delta^{\mathcal{I}}$ with the requirement that $\sigma(a) = a^{\mathcal{I}}$ if $a \in \mathcal{V}_{ind}$ or $a \in \mathcal{V}_{lit}$. The interpretation $\mathcal{I}$ satisfies a semi-ground query atom q w.r.t. σ (denoted as $\mathcal{I} \models_{\sigma} q$) if q is compatible with $\mathcal{V}_{\mathcal{O}}$ and the corresponding condition in Table 2 is satisfied. Note that, the query atoms ObjectProperty and DatatypeProperty are not in this table because they are satisfied by every interpretation as long as they are compatible with $\mathcal{V}_{\mathcal{O}}$.

The interpretation I satisfies a query $Q = q_1 \wedge \ldots \wedge q_n$ w.r.t. an evaluation σ (written $\mathcal{I} \models_{\sigma} q$) iff $\mathcal{I} \models_{\sigma} q_i$ for every $i = 1, \ldots, n$. Note that, we are only interested in the existence of an evaluation and we simply say that $\mathcal{I}$ satisfies a query Q (written $\mathcal{I} \models Q$) if there exists an evaluation σ such that $\mathcal{I} \models_{\sigma} Q$. Finally, we say that Q is a logical consequence of the ontology $\mathcal{O}$ (written $\mathcal{O} \models Q$) if the query is satisfied by every model of $\mathcal{O}$, i.e. $\mathcal{I} \models \mathcal{O}$ implies $\mathcal{I} \models Q$.

A solution to a SPARQL-DL query $Q = q_1 \wedge \ldots \wedge q_n$ w.r.t. an OWL-DL ontology $\mathcal{O}$ is a *variable mapping* $\mu : \mathcal{V}_{var} \to \mathcal{V}_{uri} \cup \mathcal{V}_{lit}$ such that when all the variables in Q

Form of the query atom	**Condition on interpretation**
$\mathsf{Type}(a, C)$	$\sigma(a) \in C^{\mathcal{I}}$
$\mathsf{PropertyValue}(a, p, v)$	$\langle \sigma(a), \sigma(v) \rangle \in p^{\mathcal{I}}$
$\mathsf{SameAs}(a, b)$	$\sigma(a) = \sigma(b)$
$\mathsf{DifferentFrom}(a, b)$	$\sigma(a) \neq \sigma(b)$
$\mathsf{SubClassOf}(C_1, C_2)$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
$\mathsf{EquivalentClass}(C_1, C_2)$	$C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$
$\mathsf{DisjointWith}(C_1, C_2)$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} = \emptyset$
$\mathsf{ComplementOf}(C_1, C_2)$	$C_1^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C_2^{\mathcal{I}}$
$\mathsf{SubPropertyOf}(p, q)$	$p^{\mathcal{I}} \subseteq q^{\mathcal{I}}$
$\mathsf{EquivalentProperty}(p, q)$	$p^{\mathcal{I}} = q^{\mathcal{I}}$
$\mathsf{Functional}(p)$	$\langle x, y \rangle \in p^{\mathcal{I}}$ and $\langle x, z \rangle \in p^{\mathcal{I}}$ implies $y = z$
$\mathsf{InverseFunctional}(p)$	$\langle y, x \rangle \in p^{\mathcal{I}}$ and $\langle z, x \rangle \in p^{\mathcal{I}}$ implies $y = z$
$\mathsf{Transitive}(p)$	$\langle x, y \rangle \in p^{\mathcal{I}}$ and $\langle y, z \rangle \in p^{\mathcal{I}}$ implies $\langle x, z \rangle \in p^{\mathcal{I}}$
$\mathsf{Symmetric}(p)$	$\langle x, y \rangle \in p^{\mathcal{I}}$ implies $\langle y, x \rangle \in p^{\mathcal{I}}$
$\mathsf{Annotation}(s, p_a, o)$	$\langle s, o \rangle \in p_a^{\mathcal{I}}$

Table 2. Satisfaction of a SPARQL-DL query atom w.r.t. an interpretation

are substituted with the corresponding value from μ we get a semi-ground query $\mu(Q)$ compatible with $\mathcal{V}_\mathcal{O}$ and $\mathcal{O} \models \mu(Q)$. The solution set $S(Q)$ for a query Q is the set of all such solutions.

4.3 From Abstract Syntax to RDF Graphs

The translation from SPARQL-DL abstract syntax to RDF triples form (which is used in the SPARQL specification) is done in a similar fashion to the translation of OWL-DL ontologies to RDF graphs. We use the function **T** from [4] that maps complex class expressions to one or more RDF triples. Anything other than complex class expressions (URIs, bnodes, literals, variables) will be mapped to itself without any additional triples, i.e. $\mathbf{T}(x) = \emptyset$. Table 3 shows the resulting set of RDF triples from translating a SPARQL-DL atom. When the transformation of a component is used as the subject, predicate, or object of a triple, that transformation is part of the result and the main node of that transformation (as defined in [4]) should be used in the triple. The translation of a SPARQL-DL query is simply the union of the RDF triples generated from translating each atom in the query. We say that any SPARQL BGP is well-formed for SPARQL-DL if it is equal to the transformation of a SPARQL-DL query in abstract syntax form.

Query Atom in Abstract Syntax	Translation to RDF Graph Form
Type(a, C)	$\langle a,$ rdf:type, $\mathbf{T}(C)\rangle$
PropertyValue(a, p, v)	$\langle a,\ p,\ v\rangle$
SameAs(a, b)	$\langle a,$ owl:sameAs, $b\rangle$
DifferentFrom(a, b)	$\langle a,$ owl:differentFrom, $b\rangle$
SubClassOf(C_1, C_2)	$\langle \mathbf{T}(C_1),$ rdfs:subClassOf, $\mathbf{T}(C_2)\rangle$
EquivalentClass(C_1, C_2)	$\langle \mathbf{T}(C_1),$ owl:equivalentClass, $\mathbf{T}(C_2)\rangle$
DisjointWith(C_1, C_2)	$\langle \mathbf{T}(C_1),$ owl:disjointWith, $\mathbf{T}(C_2)\rangle$
ComplementOf(C_1, C_2)	$\langle \mathbf{T}(C_1),$ owl:complementOf, $\mathbf{T}(C_2)\rangle$
SubPropertyOf(p, q)	$\langle p,$ rdfs:subPropertyOf, $q\rangle$
EquivalentProperty(p, q)	$\langle p,$ owl:equivalentProperty, $q\rangle$
ObjectProperty(p)	$\langle p,$ rdf:type, owl:ObjectProperty$\rangle$
DatatypeProperty(p)	$\langle p,$ rdf:type, owl:DatatypeProperty$\rangle$
Functional(p)	$\langle p,$ rdf:type, owl:FunctionalProperty$\rangle$
InverseFunctional(p)	$\langle p,$ rdf:type, owl:InverseFunctional$\rangle$
Transitive(p)	$\langle p,$ rdf:type, owl:TransitiveProperty$\rangle$
Symmetric(p)	$\langle p,$ rdf:type, owl:SymmetricProperty$\rangle$
Annotation(s, p_a, o)	$\langle s,\ p_a,\ o\rangle$

Table 3. Mapping from abstract SPARQL-DL syntax to RDF triples

5 Extensions to SPARQL-DL

There are various possibilities to extend our definition of SPARQL-DL and relax some of the restrictions. In what follows, we will first present some straightforward extensions and then mention some non-trivial extensions that cause semantic or practical problems.

5.1 Querying the Class/Property Hierarchy

One common task for ontology-based applications is to compute the class hierarchy using a reasoner. To make this task easier, DL reasoners provide API functions to get the *direct subclasses* of a class. A direct subclass is a subclass that would appear as a direct child in the hierarchy tree. Without this function one would require ad hoc code to manipulate the results from several different subclass queries. One other common query type is to get the *strict subclasses* of a given class, i.e. subclasses that are not equivalent to the given class. With standard SPARQL queries, such a query would also require one to first retrieve all subclasses, then all equivalent classes, and remove the second result set from the first result set.

In the next table, we show three additional query atoms StrictSubClassOf, DirectSubClassOf and DirectType and define their semantics based on the entailment of our original query atoms.

Query atom	**Semantics based on entailment**
$\mathcal{O} \models \mathsf{StrictSubClassOf}(C_1, C_2)$	$\mathcal{O} \models \mathsf{SubClassOf}(C_1, C_2)$ and $\mathcal{O} \not\models \mathsf{SubClassOf}(C_2, C_1)$
$\mathcal{O} \models \mathsf{DirectSubClassOf}(C_1, C_2)$	$\mathcal{O} \models \mathsf{StrictSubClassOf}(C_1, C_2)$ and $\nexists C' \in \mathcal{V}_{cls}$ s.t. $\mathcal{O} \models \mathsf{StrictSubClassOf}(C_1, C')$ and $\mathcal{O} \models \mathsf{StrictSubClassOf}(C', C_2)$
$\mathcal{O} \models \mathsf{DirectType}(a, C)$	$\mathcal{O} \models \mathsf{Type}(a, C)$ and $\nexists C' \in \mathcal{V}_{cls}$ s.t. $\mathcal{O} \models \mathsf{Type}(a, C')$ and $\mathcal{O} \models \mathsf{StrictSubClassOf}(C', C)$

Note that, the semantics for these atoms are non-monotonic because they require a query atom *not* to be entailed (which is not same as the entailment of that atom's negation). Therefore, unlike any other query atom we described so far, adding a new axiom to the KB could invalidate the results for these atoms.

We can now retrieve the whole class hierarchy by simply executing a query with the single atom $\mathsf{DirectSubClassOf}(?C_1, ?C_2)$ and then process the results starting from the result where C_2 is mapped to owl:Thing. Note that similar operators can easily be defined for super classes (DirectSuperClass) and property hierarchies (DirectSubPropertyOf and DirectSuperPropertyOf) but we leve their definitions out due to space limitations.

5.2 Supporting OWL 1.1

Our formulation of SPARQL-DL is not strongly tied to any feature in OWL-DL and the definitions can easily be migrated to OWL 1.1. For supporting OWL 1.1 in the real sense, one would like to have more query atoms specific to OWL 1.1 features, e.g. $\mathsf{Reflexive}(p)$ to test for the reflexivity of a property.

Unlike OWL-DL, there is no vocabulary separation restriction in OWL 1.1. Using punning semantics, it is possible to use the same URI to denote multiple entities. This means that we need to make the type of properties explicit in our query atoms as it is done in the OWL 1.1 abstract syntax, e.g. use SubObjectPropertyOf and SubDataPropertyOf instead of SubPropertyOf. This is simply because if a URI ex:p is declared both as an object and a datatype property, the result of SubObjectPropertyOf(?x, ex:p) will probably be different from the results of SubDataPropertyOf(?x, ex:p).

5.3 Non-trivial Extensions

Allowing Variables in Different Positions Our formulation of SPARQL-DL restricts where variables can occur. For example, we do not allow variables inside complex concept expressions; so one cannot use a query atom SubClassOf(owl:Thing, $all(?p,$ ex:C$)$) to find all the properties whose range is the class ex:C. This would certainly let one ask more powerful queries and it is still easy to define the semantics for these atoms (as long as the we keep the restriction that variables can be bound to only named entities). One might even want to use a variable in the cardinality position, e.g. SubClassOf(ex:C, $min(?n,$ ex:p, owl:Thing$)$) would return the value of the cardinality restriction on a specific property. However, this relaxation would make query evaluation much harder as it would not be possible to reduce query answering to standard reasoning services. To keep the language simple and easy to implement we excluded this possibility.

Concept Expressions in Results In SPARQL-DL, a query result maps variables to constants (named entities or literals). However, there are cases where one would like to see arbitrary class expressions in query results. An example query is SubClassOf$(C_1, ?C)$, SubClassOf$(C_2, ?C)$ which asks for the common subsumers of classes C_1 and C_2. However, without a clear definition of what kind of expressions can be included in the results, a reasoner might return infinitely many results (by generating conjunctions, disjunctions and nested value restrictions). We would also have problems encoding these results in standard SPARQL results format because variables are mapped to constants. One possibility is to define special purpose query atoms such as LeastCommonSubsumer(C_1, C_2, LCS) for which class expressions are allowed in results. The results for these queries would be returned in a free-form style as in DESCRIBE queries of SPARQL since we cannot know a priori what kind of class expressions will be returned.

Existential Variables (Bnodes) in Results As we explained earlier, bnodes in queries correspond to non-distinguished variables. For example, if we have an ontology with the assertion Type(ex:a, $some$(ex:p, ex:C)), the query Q_1 = PropertyValue$(?x, ?y)$ will not return any results because we do not know the name of the individual that variable $?y$ will be mapped to. On the other hand, the query Q_2 = PropertyValue($?x$, _:y) would be successful. One might relax the restriction in SPARQL-DL and let variables to be mapped to bnodes in the results. Then the result for Q_1 would map the variable y to a unique bnode identifier, say _:b. One might even let these bnode identifiers to be used in subsequent queries; so, for example, the query Q_3 = Type(_:b, $?C$) would return the answer ex:C. However, such an extension creates many practical problems because for each bnode identifier used in the query the reasoner would need to determine if it refers to a previously returned result or an arbitrary bnode identifier. It would also be difficult to avoid clashes for bnode identifiers in a distributed environment such as the Web.

6 Implementation

We have implemented a prototype SPARQL-DL query engine by extending Pellet's optimized ABox query engine [7]. We also support the extensions described in Section 5.1. The SPARQL-DL query engine works by first evaluating the TBox and RBox related atoms in the query (if there are any). We substitute the class or property variables using these solutions and reduce the rest of the query to a set of standard ABox queries. Answering ABox queries that contain cycles of non-distinguished variables w.r.t. an OWL-DL ontology is an open research problem and is not currently supported by Pellet. Other types of queries with non-distinguished variables are answered with the standard rolling-up technique [8].

Our implementation is still in a preliminary stage and since there are no standard benchmark problems for the mixed TBox/RBox/ABox queries we are considering, we do not yet have any performance evaluation of the overall system. But it clear that a query such as $\mathsf{Type}(?x, ?C)$ asked against a very large ABox is not practical since answering this query would require the reasoner to do instance retrieval for every class in the ontology (or realize the whole ontology which is equivalently impractical).

Our query engine can only answer SPARQL-DL subset of SPARQL BGPs. Our plan is to integrate Pellet query engine with a query engine that handles the SPARQL algebra. This way, the SPARQL engine can use the results Pellet generates for BGPs to answer more complex queries that involve OPTIONAL, UNION, or FILTER constructs.

7 Conclusions

In this paper, we have presented SPARQL-DL as a query language for OWL-DL ontologies. SPARQL-DL is a step between RDF QLs that are too unstructured w.r.t. OWL-DL and DL QLs which are not as expressive. We believe SPARQL-DL would help interoperability on the Semantic Web as it bridges this gap. As part of future work, we intend to investigate other possible extensions to SPARQL-DL including (but not limited to) aggregation operators, epistemic operators (and negation as failure), and regular expressions on OWL properties.

References

1. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C Working Draft http://www.w3.org/TR/rdf-sparql-query/ (2006)
2. Bechhofer, S., Möller, R., Crowther, P.: The DIG description logic interface. In: Proc. of the Int. Description Logics Workshop (DL 2003). (2003)
3. Haarslev, V., Moller, R., Wessel, M.: Querying the Semantic Web with Racer+ nRQL. In: Proc. of the KI-04 Workshop on Applications of Description Logics. (2004)
4. Patel-Schneider, P.F., Hayes, P., Horrocks, I.: OWL Semantics and Abstract Syntax. W3C Recommendation http://www.w3.org/TR/owl-semantics/ (2004)
5. Hayes, P.: RDF semantics. W3C Recommendation http://www.w3.org/TR/rdf-mt/ (2004)
6. Perez, J., Arenas, M., Gutierrez, C.: The semantics and complexity of SPARQL. In: 5th International Semantic Web Conference (ISWC 2006). (2006)
7. Sirin, E., Parsia, B.: Optimizations for answering conjunctive abox queries. In: Proceedings of the International Workshop on Description Logic (DL-2006). (2006)
8. Horrocks, I., Tessaris, S.: Querying the semantic web: A formal approach. In: Proc. of the 13th Int. Semantic Web Conf. (ISWC 2002). (2002) 177–191