SPARQL Beyond Subgraph Matching

Birte Glimm and Markus Krötzsch

Oxford University Computing Laboratory, UK

Abstract. We extend the Semantic Web query language SPARQL by defining the semantics of SPARQL queries under the entailment regimes of RDF, RDFS, and OWL. The proposed extensions are part of the SPARQL 1.1 Entailment Regimes working draft which is currently being developed as part of the W3C standardization process of SPARQL 1.1. We review the conditions that SPARQL imposes on such extensions, discuss the practical difficulties of this task, and explicate the design choices underlying our proposals. In addition, we include an overview of current implementations and their underlying techniques.

1 Introduction

SPARQL provides a query language for querying RDF data that has gained significant popularity since its standardization by the World Wide Consortium (W3C) in January 2008 [12]. Almost all RDF stores support SPARQL either directly or via dedicated SPARQL wrappers. The main mechanism for computing query results in SPARQL is subgraph matching: RDF triples in both the queried RDF data and the query pattern are interpreted as nodes and edges of directed graphs, and the resulting query graph is matched to the data graph using variables as wild cards.

Various W3C standards, including RDF [3] and OWL [9], provide semantic interpretations for RDF graphs that allow additional RDF statements to be inferred from explicitly given assertions. It is desirable to utilize SPARQL as a query language in these cases as well, but this requires basic graph pattern matching to be defined using semantic entailment relations instead of explicitly given graph structures. Such extensions of the SPARQL semantics are known as *entailment regimes*.

The subject of this paper is to introduce SPARQL entailment regimes for RDF and RDFS entailment [3], OWL Direct Semantics [7], and OWL RDF-Based Semantics [14]. The proposed extensions are part of the SPARQL 1.1 Entailment Regimes specification, which is currently being developed by the W3C SPARQL working group.[1] The goal of this paper is to provide a detailed outline of these proposals that is valuable to practitioners and researchers alike. We provide extended discussions of the considerations that have led to our design, and we survey principal implementation techniques.

Although SPARQL has been designed to allow for the definition of entailment regimes, their precise definition is not straightforward. Naive approaches easily lead to infinite query results that are of no practical interest. Possible reasons include trivial renamings of blank nodes, RDFS's infinitely many axiomatic triples, and the entailment of arbitrary consequences from inconsistent inputs, each of which suggests different

[1] http://www.w3.org/2009/sparql/wiki/

handling as discussed below. A second problem is that OWL is not primarily based on RDF triples but defines entailments in terms of ontological objects. Thus, triples can be genuine input data or merely part of the encoding of a complex object.

The paper is structured as follows. Section 2 gives a short introduction to RDF(S) and OWL, and Section 3 reviews the basics of SPARQL subgraph matching. In Section 4, we offer our interpretation of the conditions that SPARQL 1.0 defines for entailment regimes. The entailment regimes for RDF and RDFS are defined in Section 5, and the extensions of SPARQL with OWL's RDF-Based Semantics and the OWL Direct Semantics are presented in Section 6. Finally, Sections 7 and 8 explain basic implementation techniques for SPARQL entailment regimes and discuss further related work.

2 RDF Graphs and Their Semantics

SPARQL queries are evaluated over RDF graphs which remain the basic data structure even when adopting a more elaborate semantic interpretation. RDF is based on the set I of all *International Resource Identifiers* (IRIs), the set $\mathsf{RDF\text{-}L}$ of all *RDF literals*, and the set $\mathsf{RDF\text{-}B}$ of all *blank nodes*. The set $\mathsf{RDF\text{-}T}$ of *RDF terms* is $\mathsf{I} \cup \mathsf{RDF\text{-}L} \cup \mathsf{RDF\text{-}B}$. We generally abbreviate IRIs using prefixes rdf, rdfs, owl, and xsd to refer to the RDF, RDFS, OWL, and XML Schema Datatypes namespaces, respectively. The prefix ex is used for an imaginary example namespace.

An *RDF graph* is a set of *RDF triples* of the form $(\mathsf{subject}, \mathsf{predicate}, \mathsf{object}) \in (\mathsf{I} \cup \mathsf{RDF\text{-}B}) \times \mathsf{I} \times \mathsf{RDF\text{-}T}$. We normally omit "RDF" in our terminology if no confusion is likely, and we use Turtle syntax [1] for all examples. The *vocabulary* $\mathsf{Voc}(\mathsf{G})$ of a graph G is the set of all terms that occur in G.

Semantically, RDF graphs can be interpreted in a number of ways based on various W3C recommendations. The *simple semantics* [3] considers only the graph structure of RDF, whereas more elaborate semantics such as RDFS entailment [3] or OWL Direct Semantics [7] provide a special meaning to certain RDF terms.

The common basis for all such semantics is that they were specified by defining a model theory: one defines a suitable kind of *interpretation*, and specifies necessary and sufficient conditions for one such interpretation to *satisfy* a given RDF graph. When defining a semantics E (such as RDF, RDFS, etc.) one often speaks of E-interpretations and E-satisfaction. The set of all E-interpretations that E-satisfy a graph G are called the E-*models* of G. Semantic entailment follows from this notion: a graph G E-*entails* a graph G', written $\mathsf{G} \models_{\mathsf{E}} \mathsf{G}'$, if and only if every E-model of G is also an E-model of G'.

In this work, we encounter the *simple semantics*, *RDF semantics*, and *RDFS semantics* [3], as well as the *OWL Direct Semantics* [7] and *OWL RDF-Based Semantics* [14]. This order roughly mirrors the amount of entailments obtained under each of these semantics, e.g., all RDF-entailments are also RDFS-entailments. This ideal compatibility is not always given, especially since the OWL Direct Semantics is defined in the tradition of first-order logic, whereas the other semantics are based on a specific notion of interpretation introduced for RDF. The latter was found difficult to extend to expressive languages like OWL, and indeed entailment under the OWL RDF-Based Semantics is undecidable and is mostly used by tools that restrict to a sub-language of OWL.

On the other hand, the OWL Direct Semantics is only defined for graphs that respect certain additional conditions. This is so since this semantics is defined based on OWL objects of which RDF graphs are but an indirect representation. The OWL 2 functional-style syntax (FSS) directly corresponds to the OWL objects [8]. For example, the triple

ex:a owl:sameAs ex:b corresponds to SameIndividual(ex:a ex:b).

Since the mapping from RDF triples to OWL objects is not defined for arbitrary RDF graphs, the OWL 2 Direct Semantics makes restrictions on the well-formedness of RDF graphs that can be used with the semantics. *OWL 2 DL* describes the largest subset of RDF graphs for which the OWL 2 Direct Semantics is defined.

3 The SPARQL Query Language

We do not recall the complete surface syntax of SPARQL here but simply introduce the underlying algebraic operations using our notation. A detailed introduction to the relationship of SPARQL queries and their algebra is given in [4].

Queries are built using a countably infinite set V of *query variables* disjoint from $\mathsf{RDF\text{-}T}$. SPARQL supports a variety of *filter expressions*, or just *filters*, built from RDF terms, variables, and a number of built-in functions and operators; see [12] for details.

Definition 1. *A* triple pattern *is member of the set* $(\mathsf{RDF\text{-}T} \cup \mathsf{V}) \times (\mathsf{I} \cup \mathsf{V}) \times (\mathsf{RDF\text{-}T} \cup \mathsf{V})$, *and a* basic graph pattern *(BGP) is a set of triple patterns. More complex* graph patterns *are inductively defined to be of the form* BGP, $\mathsf{Join}(\mathsf{GP}_1, \mathsf{GP}_2)$, $\mathsf{Union}(\mathsf{GP}_1, \mathsf{GP}_2)$, $\mathsf{LeftJoin}(\mathsf{GP}_1, \mathsf{GP}_2, F)$, *and* $\mathsf{Filter}(\mathsf{F}, \mathsf{GP})$, *where* BGP *is a BGP,* F *is a filter, and* $\mathsf{GP}_{(i)}$ *are graph patterns that share no blank nodes.*[2] *The sets of* variables *and* blank nodes *in a graph pattern* GP *are denoted by* $\mathsf{V}(\mathsf{GP})$ *and* $\mathsf{B}(\mathsf{GP})$, *respectively.*

SPARQL allows literals to be used as triple subjects although RDF graphs cannot currently contain such triples. This is meant to support (future) extensions of RDF.

We exclude a number of SPARQL features from our discussion. First, we disregard any of the new SPARQL 1.1 query constructs since their syntax and semantics are still under discussion in the SPARQL working group. Second, we do not consider output formats (e.g., SELECT or CONSTRUCT) and solution modifiers (e.g., LIMIT or OFFSET) which are not affected by entailment regimes. Third, we exclude SPARQL datasets that allow SPARQL endpoints to cluster data into several named graphs and a default graph. For simpler presentation, we omit dataset clauses and assume that queries are evaluated over the default graph, called the *active graph* for the query.

Evaluating a SPARQL graph pattern results in a *solution sequence* that lists possible bindings of query variables to RDF terms in the active graph. Such bindings are represented by partial functions μ from V to $\mathsf{RDF\text{-}T}$, called *solution mappings*. For a solution mapping μ – and more generally for any (partial) function – the set of elements on which μ is defined is the *domain* $\mathsf{dom}(\mu)$ of μ, and the set $\mathsf{ran}(\mu) := \{\mu(x) \mid x \in \mathsf{dom}(\mu)\}$ is the *range* of μ. For a graph pattern GP, we use $\mu(\mathsf{GP})$ to denote the pattern obtained by

[2] As in [12], disallowing GP_1 and GP_2 to share blank nodes is important to avoid unintended co-references. This was not needed in [10] where blank nodes were not considered.

Table 1. Evaluation of algebraic operators in SPARQL

$$
\begin{aligned}
[\![\mathsf{Union}(\mathsf{GP}_1,\mathsf{GP}_2)]\!]_\mathsf{G} &:= \big\{(\mu,n) \mid n = M_1(\mu) + M_2(\mu) > 0\big\}\\
[\![\mathsf{Join}(\mathsf{GP}_1,\mathsf{GP}_2)]\!]_\mathsf{G} &:= \big\{(\mu,n) \mid n = \textstyle\sum_{(\mu_1,\mu_2)\in J(\mu)} (M_1(\mu_1) * M_2(\mu_2)) > 0\big\} \text{ where}\\
&\ J(\mu) := \{(\mu_1,\mu_2) \mid \mu_1,\mu_2 \text{ compatible and } \mu = \mu_1 \cup \mu_2\}\\
[\![\mathsf{Filter}(\mathsf{F},\mathsf{GP})]\!]_\mathsf{G} &:= \big\{(\mu,n) \mid M(\mu) = n > 0 \text{ and } [\![\mu(\mathsf{F})]\!] = \mathsf{true}\big\}\\
[\![\mathsf{LeftJoin}(\mathsf{GP}_1,\mathsf{GP}_2,\mathsf{F})]\!]_\mathsf{G} &:= [\![\mathsf{Filter}(\mathsf{F},\mathsf{Join}(\mathsf{GP}_1,\mathsf{GP}_2))]\!]_\mathsf{G}\ \cup\\
&\ \big\{(\mu_1, M_1(\mu_1)) \mid \text{for all } \mu_2 \text{ with } M_2(\mu_2) > 0 : \mu_1 \text{ and } \mu_2 \text{ are}\\
&\qquad\qquad \text{incompatible or } [\![(\mu_1 \cup \mu_2)(\mathsf{F})]\!] = \mathsf{false}\big\}
\end{aligned}
$$

applying μ to all elements of GP in $\mathsf{dom}(\mu)$. This convention is extended in the obvious way to filter expressions, and to all functions that are defined on variables or terms.

The order of solution sequences is relevant for later processing steps in SPARQL, but not for obtaining the solutions for a graph pattern. To disregard the order formally, we use *solution multisets*. A *multiset* over an *underlying set S* is a total function $M\colon S \to \mathbb{N}^+ \cup \{\omega\}$ where $\mathbb{N}^+$ are the positive natural numbers, and $\omega > n$ for all $n \in \mathbb{N}^+$. The value $M(s)$ is the *multiplicity* of $s \in S$, and ω denotes a countably infinite number of occurrences. Infinitely many occurrences of individual solution mappings are indeed possible when considering SPARQL entailment regimes, and a major concern of this work is to avoid this for the entailment regimes we define.

We often represent a multiset M with underlying set S by the set $\{(s, M(s)) \mid s \in S\}$. Accordingly, we may write $(s,n) \in M$ if $M(s) = n$. Also, we assume that $M(s)$ denotes 0 whenever $s \notin S$. In some cases, it is also convenient to use a set-like notation where repeated elements are allowed, e.g. writing $\dot{\{}a, b, b\dot{\}}$ for the multiset M with underlying set $\{a,b\}$, $M(a) = 1$, and $M(b) = 2$.

To define the solution multiset for a BGP under the simple semantics, we still need to consider the effect of blank nodes. Intuitively, these act like variables that are projected out of a query result, and thus they may lead to duplicate solution mappings. This is accounted for using RDF instance mappings as follows:

Definition 2. *An* RDF instance mapping *is a partial function $\sigma\colon \mathsf{RDF\text{-}B} \to \mathsf{RDF\text{-}T}$ from blank nodes to RDF terms. We extend σ to pattern graphs and filters as done for solution mappings above. The* solution multiset $[\![\mathsf{BGP}]\!]_\mathsf{G}$ *for a basic graph pattern* BGP *over the active graph* G *is the following multiset of solution mappings:*

$$
\begin{aligned}
\{(\mu,n) \mid\ & \mathsf{dom}(\mu) = \mathsf{V}(\mathsf{BGP}), \textit{ and } n \textit{ is the maximal number such that}\\
& \sigma_1,\ldots,\sigma_n \textit{ are distinct RDF instance mappings such that, for all } 1 \le i \le n,\\
& \mathsf{dom}(\sigma_i) = \mathsf{B}(\mathsf{BGP}) \textit{ and } \mu(\sigma_i(\mathsf{BGP})) \textit{ is a subgraph of } \mathsf{G}\}.
\end{aligned}
$$

Note that the number n in the definition of $[\![\mathsf{BGP}]\!]_\mathsf{G}$ is always finite.

The algebraic operators that are required for evaluating non-basic graph patterns correspond to operations on multisets of solution mappings which are the same for all entailment regimes. To take infinite multiplicities into account, we assume $\omega + n = n + \omega = \omega$ for all $n \ge 0$, $\omega * n = n * \omega = \omega$ for all $n > 0$ and $\omega * 0 = 0 * \omega = 0$. To

Table 2. Conditions for extending BGP matching to E-entailment (quoted from [12])

1. The scoping graph SG, corresponding to any consistent active graph AG, is uniquely specified and is E-equivalent to AG.
2. For any basic graph pattern BGP and pattern solution mapping P, P(BGP) is well-formed for E.
3. For any scoping graph SG and answer set $\{P_1, \ldots, P_n\}$ for a basic graph pattern BGP, and where $BGP_1, \ldots, BGP_n$ is a set of basic graph patterns all equivalent to BGP, none of which share any blank nodes with any other or with SG
$$SG \models_E (SG \cup P_1(BGP_1) \cup \ldots \cup P_n(BGP_n)).$$
4. Each SPARQL extension must provide conditions on answer sets which guarantee that every BGP and AG has a finite set of answers which is unique up to RDF graph equivalence.

incorporate the effect of filters, it suffices to know that SPARQL assigns to any filter F an effective truth value that we will denote by $[\![F]\!]$.

Definition 3. *Two solution mappings μ_1 and μ_2 are* compatible *if $\mu_1(v) = \mu_2(v)$ for all $v \in \mathsf{dom}(\mu_1) \cap \mathsf{dom}(\mu_2)$. If this is the case, a solution mapping $\mu_1 \cup \mu_2$ is defined by setting $(\mu_1 \cup \mu_2)(v) := \mu_1(v)$ if $v \in \mathsf{dom}(\mu_1)$, and $(\mu_1 \cup \mu_2)(v) := \mu_2(v)$ otherwise.*

The evaluation *of a graph pattern over G, denoted $[\![\cdot]\!]_G$, is defined as in Table 1, where we abbreviate multisets $[\![GP]\!]_G / [\![GP_1]\!]_G / [\![GP_2]\!]_G$ by $M / M_1 / M_2$ for readability.*

Note that two mappings with disjoint domains are always compatible. Intuitively, $\mathsf{Join}(GP_1, GP_2)$ represents all possible combinations of mappings from $[\![GP_1]\!]_G$ with compatible mappings from $[\![GP_2]\!]_G$, as accounted for by taking the product of multiplicities. One mapping in a join may result from various combinations of compatible mappings, so that we need to compute a sum of their multiplicities. The expression $\mathsf{LeftJoin}(GP_1, GP_2, F)$ combines the filtered join of the inputs with all mappings of $[\![GP_1]\!]_G$ which are not represented in this filtered join.

4 Extending Basic Graph Pattern Matching

To extend SPARQL for entailment regimes like RDFS or OWL Direct Semantics, it suffices to modify the evaluation of BGPs accordingly, while the remaining algebra operations can still be evaluated as in Definition 3. When considering E-entailment, we thus define solution multisets $[\![BGP]\!]^E_G$. The SPARQL Query 1.0 specification [12] already envisages the extension of the BGP matching mechanism, and provides a set of conditions for such extensions that we recall in Table 2. We found these conditions hard to interpret since their terminology is not aligned well with the remaining specification. In the following, we discuss our reading of these conditions, leading to a revised clarified version presented in Table 3.[3]

[3] The current SPARQL working group is not chartered to revise the existing specification, so the ongoing work on entailment regimes is based on the assumption that the conditions were meant to be in the revised form.

Table 3. Clarified conditions for extending BGP matching to E-entailment

An entailment regime E provides conditions on BGP evaluation such that for any evaluation $[\![\cdot]\!]^{\mathsf{E}}_{\mathsf{G}}$ that satisfies these conditions, any basic graph pattern BGP, and any graph G, the multiset of graphs $\{(\mu(\mathsf{BGP}), n) \mid (\mu, n) \in [\![\mathsf{BGP}]\!]^{\mathsf{E}}_{\mathsf{G}}\}$ is uniquely determined up to RDF graph equivalence.

1. For any consistent active graph AG, the entailment regime E uniquely specifies a *scoping graph* SG that is E-equivalent to AG.
2. A set of *well-formed* graphs for E is specified such that, for any basic graph pattern BGP, scoping graph SG, and solution mapping μ in the underlying set of $[\![\mathsf{BGP}]\!]^{\mathsf{E}}_{\mathsf{SG}}$, the graph $\mu(\mathsf{BGP})$ is well-formed for E.
3. For any basic graph pattern BGP, and scoping graph SG, if S denotes the underlying set of $[\![\mathsf{BGP}]\!]^{\mathsf{E}}_{\mathsf{SG}}$, then there is a family of RDF instance mappings $(\sigma_\mu)_{\mu \in S}$ such that
$$\mathsf{SG} \models_{\mathsf{E}} \mathsf{SG} \cup \bigcup_{\mu \in S} \mu(\sigma_\mu(\mathsf{BGP})).$$
4. Entailment regimes *should* provide conditions to prevent trivial infinite solution multisets.

Condition (1) forces an entailment regime to specify a so-called *scoping graph* based on which query answers are computed instead of using the active graph directly. Since an entailment regime's definition of BGP matching is free to refer to such derived graph structures anyway, the additional use of a scoping graph does not increase the freedom of potential extensions. We assume, therefore, that the scoping graph is the active graph in the remainder. If the active graph is E-inconsistent, entailment regimes specify the intended behavior directly, e.g., by requiring that an error is reported.

Condition (2) refers to a "pattern solution mapping" though what is probably meant is a *pattern instance mapping* P, defined in [12] as the combination of an RDF instance mapping σ and a solution mapping μ where $\mathsf{P}(x) = \mu(\sigma(x))$. We assume, however, that (2) is actually meant to refer to all solution mappings in $[\![\mathsf{BGP}]\!]^{\mathsf{E}}_{\mathsf{G}}$. Indeed, even for simple entailment where well-formedness only requires $\mathsf{P}(\mathsf{BGP})$ to be an RDF graph, condition (2) would be violated when using *all* pattern instance mappings. To see this, consider a basic graph pattern $\mathsf{BGP} = \{\texttt{\_:a ex:b ex:c}\}$. Clearly, there is a pattern instance mapping P with $\mathsf{P}(\texttt{\_:a}) = \texttt{"1"\^{}\^{}xsd:int}$, but $\mathsf{P}(\mathsf{BGP}) = \{\texttt{"1"\^{}\^{}xsd:int ex:b ex:c}\}$ is not an RDF graph. Similar problems occur when using all solution mappings. Hence we assume (2) to refer to elements of the computed solution multiset $[\![\mathsf{BGP}]\!]^{\mathsf{E}}_{\mathsf{G}}$. The notion of *well-formedness* in turn needs to be specified explicitly for entailment regimes.

Condition (3) uses the term "answer set" to refer to the results computed for a BGP. To match the rest of [12], this has to be interpreted as the solution multiset $[\![\mathsf{BGP}]\!]^{\mathsf{E}}_{\mathsf{G}}$. This also means mappings P_i are solution mappings (not pattern instance mappings as their name suggests). The purpose of (3), as noted in [12], is to ensure that if blank node names are returned as bindings for a variable, then the same blank node name occurs in different solutions only if it corresponds to the same blank node in the graph. To illustrate the problem, consider the following graphs:

$$
\begin{array}{llll}
\mathsf{G} : \texttt{ex:a ex:b \_:c.} & \mathsf{G}_1 : \texttt{ex:a ex:b \_:b}_1. & \mathsf{G}_2 : \texttt{ex:a ex:b \_:b}_2. & \mathsf{G}_3 : \texttt{ex:a ex:b \_:b}_1. \\
\quad \texttt{\_:d ex:e ex:f.} & \quad \texttt{\_:b}_2 \texttt{ ex:e ex:f.} & \quad \texttt{\_:b}_1 \texttt{ ex:e ex:f.} & \quad \texttt{\_:b}_1 \texttt{ ex:e ex:f.}
\end{array}
$$

Clearly, G simply entails G_1 and G_2, but not G_3 where the two blank nodes are identified. Now consider a basic graph pattern $\mathsf{BGP} = \{\mathsf{ex{:}a\ ex{:}b\ ?x.?y\ ex{:}e\ ex{:}f}\}$. A solution multiset for BGP could comprise two mappings $\mu_1 \colon \mathsf{?x} \mapsto \mathsf{\_{:}b}_1, \mathsf{?y} \mapsto \mathsf{\_{:}b}_2$ and $\mu_2 \colon \mathsf{?x} \mapsto \mathsf{\_{:}b}_2, \mathsf{?y} \mapsto \mathsf{\_{:}b}_1$. So we have $\mu_1(\mathsf{BGP}) = \mathsf{G}_1$ and $\mu_2(\mathsf{BGP}) = \mathsf{G}_2$, and both solutions are entailed. However, condition (3) requires that $\mathsf{G} \cup \mu_1(\mathsf{BGP}) \cup \mu_2(\mathsf{BGP})$ is also entailed by G, and this is not the case in our example since this union contains G_3. The reason is that our solutions have unintended co-references of blank nodes that (3) does not allow. SPARQL's basic subgraph matching semantics respects this condition by requiring solution mappings to refer to blank nodes that actually occur in the active graph, so blank nodes are treated like (Skolem) constants.[4] The revised condition in Table 3 has further been modified to not implicitly require finite solution multisets which may not be appropriate for all entailment regimes. In addition, we use RDF instance mappings for renaming blank nodes instead of requiring renamed variants of the BGP.

Finally, condition (4) requires that solution multisets are finite and uniquely determined up to RDF graph equivalence, again using the "answer set" terminology. Our revised condition clarifies what it means for a solution multiset to be "unique up to RDF graph equivalence." We move the uniqueness requirement above all other conditions, since (2) and (3) do not make sense if the solution multiset was not defined in this sense. The rest of the condition was relaxed since entailment regimes may inherently require infinite solution multisets, e.g., in the case of the Rule Interchange Format RIF [6]. It is desirable that this only happens if there are infinite solutions that are "interesting," so the condition has been weakened to merely recommend the elimination of infinitely many "trivial" solution mappings in solution multisets. The requirement thus is expressed in an informal way, leaving the details to the entailment regime. Within this paper, we will make sure that the solution multisets are in fact finite (both regarding the size of the underlying set, and regarding the multiplicity of individual elements).

5 The RDF and RDFS Entailment Regimes

We focus on specifying the RDFS entailment regime, since the case of RDF is an obvious simplification of this entailment regime. The major problem for RDFS entailment is to avoid trivially infinite solution multisets as suggested by Table 3 (4), where three principal sources of infinite query results have to be addressed:

1. An RDF graph can be inconsistent under the RDFS semantics in which case it RDFS-entails all (infinitely many) conceivable triples.
2. The RDFS semantics requires all models to satisfy an infinite number of *axiomatic triples* even when considering an empty graph.
3. Every non-empty graph entails infinitely many triples obtained by using arbitrary blank nodes in triples.

We now discuss each of these problems, and derive a concrete definition for BGP matching in the proposed entailment regime at the end of this section.

[4] Yet, SPARQL allows blank nodes to be renamed when loading documents, so there is no guarantee that blank node IDs used in input documents are preserved.

5.1 Treatment of Inconsistencies

SPARQL does not require entailment regimes to yield a particular query result in cases where the active graph is inconsistent. As stated in [12], "[the] effect of a query on an inconsistent graph [. . .] must be specified by the particular SPARQL extension." One could simply require that implementations of the RDFS entailment report an error when given an inconsistent active graph. However, a closer look reveals that inconsistencies are extremely rare in RDFS, so that the requirement of checking consistency before answering queries would impose an unnecessary burden on implementations.

Indeed, graphs can only be RDFS-inconsistent due to improper use of the datatype rdf:XMLLiteral. A typical example for this is the following graph:

ex:a ex:b "<"^^rdf:XMLLiteral.   ex:b rdfs:range rdfs:Literal.

The literal in the first triple is *ill-typed* as it does not denote a value of rdf:XMLLiteral. This does not cause an inconsistency yet but forces "<"^^rdf:XMLLiteral to be interpreted as a resource that is not in the extension of rdfs:Literal, which in turn cannot be the case in any model that satisfies the second triple. Ill-typed literals are the only possible cause of inconsistency in RDFS and as such not a frequent problem.[5] Moreover, inconsistencies of this type are inherently "local" as they are based on individual ill-typed literals that could easily be ignored if not related to a given query.

It has thus been decided in the SPARQL working group that systems only have to report an error if they actually detect an inconsistency. Until this happens, queries can be answered as if all literals were well-typed. Our exact formalization corresponds to a behavior where tools simply assume that all strings are well-typed for rdf:XMLLiteral, and hence does not put additional burden on implementers.

5.2 Treatment of Axiomatic Triples

Every RDFS model is required to satisfy an infinite number of *axiomatic triples*. The reason is that the RDF vocabulary for encoding lists includes property names rdf:_i for all $i \geq 1$, with several (RDFS) axiomatic triples for each rdf:_i. For instance, we find a triple rdf:_i rdf:type rdf:Property for all $i \in \mathbb{N}$. Thus, the query ?x rdf:type rdf:Property could have infinitely many results. We consider such results trivial in the sense of Table 3 (4), and thus we want avoid them in the RDFS entailment regime.

We therefore propose that axiomatic triples with a subject of the form rdf:_i are only taken into account if the subject's IRI explicitly occurs in the active graph. This ensures that only finitely many axiomatic triples are considered, since there is only a finite number of axiomatic triples whose subjects do not have the form rdf:_i. To conveniently formalize this, Definition 5 below still refers to the standard RDFS entailment with all axiomatic triples, and restricts the range of solution mappings to an *answer domain* instead. Ignoring axiomatic triples for IRIs rdf:_i that occur only in a query but not in the active graph ensures that the total number of entailments that are relevant for query answering is finite. This would not be the case if new entailments would be required

[5] Implementations may support additional datatypes that can lead to similar problems. Such extensions go beyond the RDFS semantics we consider here, yet inconsistencies remain rare even in these cases.

whenever a given query contains a hitherto unused IRI. This distinguishes our approach from [5] where a *partial closure* algorithm is used to decide RDFS entailment for a set of axiomatic triples based on both the given graph and the query graph.

5.3 Treatment of Blank Nodes

Even if condition (3) in Table 3 holds, solution multisets could include infinitely many results that only differ in the identifiers for blank nodes. Simple entailment avoids this problem by restricting results to blank nodes that occur in the active graph. For entailment regimes, however, one must take entailed triples into account. This already leads to triples with different blank nodes, as illustrated in the graphs G_1 and G_2 in Section 4.

Restricting the range of solution mappings to blank nodes in the active graph would ensure finiteness but is not a satisfactory solution. To see why, consider the graph

$$\mathsf{G} : \mathsf{ex{:}a\ ex{:}b\ ex{:}c.} \quad \mathsf{ex{:}d\ ex{:}e\ \_{:}f.}$$

The query $\mathsf{BGP} = \{\mathsf{ex{:}a\ ex{:}b\ ?x}\}$ yields only one solution mapping $\mu : \mathsf{?x} \mapsto \mathsf{ex{:}c}$ under simple entailment. Yet, the mapping $\mu' : \mathsf{?x} \mapsto \mathsf{\_{:}f}$ uses only blank nodes from G, and satisfies $\mathsf{G} \models \mu'(\mathsf{BGP})$ even under simple semantics. This shows that the latter two conditions are not sufficiently specific for handling blank nodes in entailment regimes. A more adequate approach is the use of *Skolemization*:

Definition 4. *Let the prefix* skol *refer to a namespace IRI that does not occur as the prefix of any IRI in the active graph or query. The* Skolemization $\mathsf{sk(\_{:}b)}$ *of a blank node* $\mathsf{\_{:}b}$ *is defined as* $\mathsf{sk(\_{:}b)} := \mathsf{skol{:}b}$. *We extend* $\mathsf{sk}(\cdot)$ *to graphs and filters just like other (partial) functions on RDF terms.*

Intuitively, Skolemization changes blank nodes into resource identifiers that are not affected by entailment. Clearly, we do not want Skolemized blank nodes to occur in query results, but it is useful to restrict to solution mappings μ for which $\mathsf{sk(G)} \models \mathsf{sk}(\mu(\mathsf{BGP}))$. In the above example, this condition is indeed satisfied by μ but not by μ'.

5.4 Defining the RDF(S) Entailment Regimes

The set of *well-formed* graphs for the RDFS entailment regime is simply the set of all RDF graphs. BGP matching for RDFS is defined as follows.

Definition 5. *Let* $\mathsf{Voc(RDFS)}$ *be the RDFS vocabulary,* G *an RDF graph, and* BGP *a basic graph pattern. The* answer domain w.r.t. G under RDFS entailment, *written* $\mathsf{AD_{RDFS}(G)}$, *is the set* $\mathsf{Voc(G)} \cup (\mathsf{Voc(RDFS)} \setminus \{\mathsf{rdf{:}\_i} \mid i \in \mathbb{N}\})$. *The* evaluation of BGP over G under RDFS entailment $[\![\mathsf{BGP}]\!]_{\mathsf{G}}^{\mathsf{RDFS}}$ *is the solution multiset*

$$\begin{aligned}\{(\mu, n) \mid\ & \mathsf{dom}(\mu) = \mathsf{V(BGP)}, \textit{and } n \textit{ is the maximal number such that} \\ & \sigma_1, \ldots, \sigma_n \textit{ are distinct RDF instance mappings such that, for each } 1 \le i \le n, \\ & \mathsf{sk(G)} \models_{\mathsf{RDFS}} \mathsf{sk}(\mu(\sigma_{\mathsf{i}}(\mathsf{BGP}))) \textit{ and } (\mathsf{ran}(\mu) \cup \mathsf{ran}(\sigma_{\mathsf{i}})) \subseteq \mathsf{AD_{RDFS}(G)}\}.\end{aligned}$$

Other types of graph patterns are evaluated as in Definition 3. If the active graph is RDFS-inconsistent, implementations may compute solution multisets based on the assumption that all literals of type rdf:XMLLiteral are well-typed, so that no inconsistency occurs. When the inconsistency is detected, implementations should report an error.

Since computing a partial RDFS closure for an RDF graph can be done in polynomial time [5] and BGP evaluation then amounts to subgraph matching over the partial closure, it follows that the complexity of the evaluation problem under the RDFS regime is the same as for standard SPARQL. For set semantics instead of multiset semantics this is known to be PSPACE-complete [10].

The entailment regime for RDF is defined similarly, but using RDF entailment and the RDF vocabulary instead. Note that the above definition can also be restricted to simple entailment, yielding the same solution multisets as Definition 2.

6 The OWL Entailment Regimes

In contrast to the RDFS semantics, a graph does no longer admit a unique canonical model that can be used to compute answers under the RDF-Based Semantics (RBS) and Direct Semantics (DS) of OWL, i.e., we can no longer imagine queries to act on a unique "completed" version of the active graph. This affects reasoning algorithms (see Section 7), but has only little effect on our definitions. The main new challenges for OWL are its expressive datatype constructs that may lead to infinite answers, and the fact that the OWL DS is defined in terms of OWL objects to which a given RDF graph and query must first be translated. The problems discussed for RDF(S) also require slightly different solutions for OWL:

1. Inconsistent input ontologies are required to be rejected with an error.
2. The axiomatic triples of RDFS are used only by the RBS and can again be handled by suitably restricting solutions to an answer domain.
3. The problem of blank nodes occurs for both semantics and can again be addressed by Skolemization, but for DS the blank nodes that are used to encode OWL objects must not be Skolemized.

The main difference to RDFS is the stricter first item which no longer permits deferred inconsistency detection. Inconsistencies in RDFS were easy to ignore since they always related to single literals. Neither OWL semantics suggests such simple reasoning under inconsistencies. Although proposals exists for addressing this, they disagree on the inferred entailments and tend to require complex computations. On the other hand, typical OWL reasoning algorithms are model building procedures which detect inconsistencies as part of their normal operation. Hence, reporting errors in this case can usually be done without additional effort.

6.1 Infinite Entailments in Datatype Reasoning

In order to see how datatype reasoning in OWL can cause infinite entailments, consider the graph and query in Table 4. Recall that a abbreviates rdf:type, [. . .] denotes an implicit blank node, and (. . .) denotes an RDF list. G states that all data values to which Peter is related via ex:dp are in the singleton set of the integer 5. The query asks for all data values to which ex:Peter cannot be related with ex:dp. Without suitable restrictions, all (infinitely many) integers other than 5 could be used in solution mappings for ?x.

Table 4. A query with infinitely many entailed solutions

```
G : ex:Peter a [ a owl:Restriction;            BGP : ex:Peter a [ a owl:Restriction;
      owl:onProperty ex:dp;                          owl:onProperty ex:dp;
      owl:allValuesFrom [ a rdfs:Datatype;           owl:allValuesFrom [ a rdfs:Datatype;
        owl:oneOf ("5"^^xsd:integer)]]                 owl:datatypeComplementOf [
                                                         a rdfs:Datatype; owl:oneOf (?x)]]]
```

Moreover, it is currently unknown how to compute all mappings for literal variables even for cases where there number is finite – testing all literals is clearly not an option.[6]

We therefore restrict the answer domain for the OWL entailment regimes to include only literals that are explicitly mentioned in the input graph. Like for the IRIs rdf:_i, this may lead to unexpected behavior, since mentioning a literal in the input may lead to new query results even for queries not directly related to this literal. Yet, we think this problem is so rare in practice that a more detailed analysis of the problematic datatype expressions is not worthwhile, even if it could further limit unintuitive behavior.

6.2 The OWL 2 RDF-Based Semantics Entailment Regime

The OWL 2 RDF-Based Semantics treats classes as individuals that refer to elements of the domain. Each such element is then associated with a subset of the domain, called the class extension. This means that semantic conditions on class extensions are only applicable to those classes that are actually represented by an element of the domain which can lead to less consequences than expected. Consider the following An example is given by the following graph and BGP:

```
G : ex:a rdf:type ex:C        BGP : ?x rdf:type [ rdf:type owl:Class ;
                                                  owl:unionOf ( ex:C ex:D ) ]
```

G states that ex:a has type ex:C, while BGP asks for instances of the complex class denoting the union of ex:C and ex:D. One might expect μ: ?x $\mapsto$ ex:a to be a solution, but this is not the case under the OWL 2 RDF-Based Semantics (see also [14, Sec. 7.1]). It is guaranteed that the union of the class extensions for ex:C and ex:D exists as a subset of the domain; no statement in G implies, however, that this union is the class extension of any domain element. Thus, μ(BGP) is not entailed by G.

The entailment holds, however, when the statement ex:E owl:unionOf (ex:C ex:D) is added to G. In the OWL Direct Semantics, in contrast, classes denote sets and not domain elements, so G entails μ(BGP) under DS where, formally, G must first be extended with an ontology header to become well-formed for DS. Note that a similar situation occurs for the example in Section 6.1, but the problem still occurs if the necessary expressions are introduced.

Summing up, the RBS handles blank nodes just like RDFS, even in cases where they are needed for encoding OWL class expressions. This allows us to use Skolemization just like in the case of RDFS in the next definition.

[6] Hence one cannot call such solutions “trivial” in the sense of Table 3. Indeed, our restrictions are motivated by pragmatic considerations, not by formal requirements of SPARQL.

Table 5. Grammar extension for extended OWL objects

Class :=	IRI | Var	ObjectProperty :=	IRI | Var	DataProperty :=	IRI | Var
Individual :=	NamedIndividual | AnonymousIndividual | Var				
Literal :=	typedLiteral | stringLiteralNoLanguage | stringLiteralWithLanguage | Var				

Definition 6. *Let* $\mathsf{Voc(OWL2)}$ *be the OWL 2 vocabulary,* G *a graph, and* BGP *a basic graph pattern. We write* $\models_{\mathsf{RBS}}$ *to denote the OWL 2 RDF-Based Semantics entailment relation. The* answer domain w.r.t. G under RDF-Based Semantics entailment, *written* $\mathsf{AD_{RBS}(G)}$*, is the set* $\mathsf{Voc(G)} \cup (\mathsf{Voc(OWL2)} \setminus \{\mathsf{rdf{:}\_i} \mid i \in \mathbb{N}\})$*. The* evaluation of BGP over G under RDF-Based Semantics entailment $[\![\mathsf{BGP}]\!]_{\mathsf{G}}^{\mathsf{RBS}}$ *is the solution multiset*

$$\{(\mu, n) \mid \mathsf{dom}(\mu) = \mathsf{V(BGP)}, \textit{ and } n \textit{ is the maximal number such that } \sigma_1, \ldots, \sigma_n \textit{ are distinct RDF instance mappings such that, for each } 1 \le i \le n, \mathsf{sk(G)} \models_{\mathsf{RBS}} \mathsf{sk}(\mu(\sigma_{\mathsf{i}}(\mathsf{BGP}))) \textit{ and } (\mathsf{ran}(\mu) \cup \mathsf{ran}(\sigma_{\mathsf{i}})) \subseteq \mathsf{AD_{RBS}(G)}\}.$$

6.3 The OWL 2 Direct Semantics Entailment Regime

The OWL 2 Direct Semantics is not defined in terms of triples, but in terms of OWL objects that constitute an *ontology*. The OWL 2 recommendation specifies how to construct an ontology $\mathsf{O_G}$ from a graph G that satisfies some further conditions [9]. Thus G is *well-formed for the OWL DS entailment regime* if $\mathsf{O_G}$ is defined. In the following, we conveniently identify ontologies with their unique canonical representation in Functional-Style Syntax [8]. Some RDF triples are mapped to so-called *non-logical axioms* such as annotations, declarations, or import directives. Such axioms can only have indirect effect on DS entailment, e.g., since imported axioms are taken into account, but they do not directly lead to entailments. In particular, annotations do not contribute query results under DS.

Like the active graph, also the BGP of the query is mapped into an OWL 2 DL ontology, extended to allow variables in place of class names, object property names, datatype property names, individual names, or literals. Table 5 shows how productions of the OWL 2 functional-style syntax grammar [8] are extended to allow variables as defined by the Var production from the SPARQL grammar [12]. Solution mappings in a query result are applied to such extended ontologies to obtain a set of OWL DL axioms that is compatible with the queried ontology and also entailed by it under DS.

The construction of ontologies from graphs requires type declarations for properties, classes, and (custom) datatypes to avoid ambiguities, and we need similar typing information for terms *and* variables in BGPs. For example, the BGP {?s ?p ?o} could refer to DataPropertyAssertion(?p ?s ?o) or ObjectPropertyAssertion(?p ?s ?o) if the type of ?p is not given. We take type declarations from the queried ontology into account, so that only variables may require further typing.

Formally, an extended ontology $\mathsf{O^G_{BGP}}$ is constructed for a basic graph pattern BGP and graph G using the parsing process for RDF graphs as defined in [9] with three modifications: variable identifiers are allowed in place of IRIs and literals in all parsing steps, an ontology header may be added to BGP if not given, and the type declarations given in BGP are augmented with the declarations in G (denoted $\mathsf{AllDecl(G)}$ in [9]). The complete parsing process is detailed in the latest entailment regimes working draft.[7] BGP is *well-formed for the OWL DS entailment regime and a graph* G if $\mathsf{O}^{\mathsf{G}}_{\mathsf{BGP}}$ can be obtained in this way and is an extended OWL DL ontology.

We can now define the evaluation of graph patterns. Skolemization is now applied to O_{G}, which ensures that only blank nodes that represent anonymous OWL individuals are Skolemized, not blank nodes used for encoding complex OWL syntax in RDF.

Definition 7. *Consider a graph* G *that is well-formed for the OWL 2 DS entailment regime, and a basic graph pattern* BGP *that is well-formed for DS and* G*. With* $\mathsf{sk}(\mathsf{O}_{\mathsf{G}})$ *we denote the result of replacing each blank node b in* O_{G} *with* $\mathsf{sk}(\mathsf{b})$*. The* answer domain w.r.t. G under OWL 2 Direct Semantics entailment*, written* $\mathsf{AD}_{\mathsf{DS}}(\mathsf{G})$*, is* $\mathsf{Voc}(\mathsf{O}_{\mathsf{G}})$*. If* O_{G} *is inconsistent, queries must be rejected with an error. Otherwise, we write* $\models_{\mathsf{DS}}$ *for the OWL 2 Direct Semantics entailment relation and define the* evaluation of BGP over G under OWL 2 Direct Semantics entailment $[\![\mathsf{BGP}]\!]^{\mathsf{DS}}_{\mathsf{G}}$ *as the solution multiset*

$$
\begin{aligned}
\{(\mu, n) \mid{} & \mathsf{dom}(\mu) = \mathsf{V}(\mathsf{BGP}), \textit{ and n is the maximal number such that} \\
& \sigma_1, \ldots, \sigma_n \textit{ are distinct RDF instance mappings such that, for each } 1 \leq i \leq n, \\
& \mathsf{O}_{\mathsf{G}} \cup \mu(\sigma_i(\mathsf{O}^{\mathsf{G}}_{\mathsf{BGP}})) \textit{ is an OWL 2 DL ontology, and} \\
& \mathsf{sk}(\mathsf{O}_{\mathsf{G}}) \models_{\mathsf{DS}} \mathsf{sk}(\mu(\sigma_{\mathsf{i}}(\mathsf{O}^{\mathsf{G}}_{\mathsf{BGP}}))) \textit{ and } (\mathsf{ran}(\mu) \cup \mathsf{ran}(\sigma_{\mathsf{i}})) \subseteq \mathsf{AD}_{\mathsf{DS}}(\mathsf{G})\}.
\end{aligned}
$$

Since $\mathsf{AD}_{\mathsf{DS}}(\mathsf{G})$ is finite, clearly the solution multiset and each multiplicity is finite too. Although the restriction to $\mathsf{AD}_{\mathsf{DS}}(\mathsf{G})$ avoids infinite results as discussed in Section 6.1, reasoners may have to consider a large number of literals as potential variable bindings and we expect that not all systems will provide a complete implementation for queries with literal variables.

The complexity of standard reasoning problems in OWL are well-understood and BGP evaluation can be implemented using the standard reasoning techniques. The complexity of OWL reasoning usually outweighs that of the SPARQL algebra operations, i.e., checking whether a solution mapping is a solution is complete for nondeterministic exponential time in OWL DL and undecidable for the RDF-Based semantics.

7 Implementations of SPARQL Entailment Regimes

We now discuss how the interplay between SPARQL query processing and semantic inference can be implemented in practice. Three principal approaches for this task are reviewed below. An overview of optimized implementation techniques for SPARQL algebra operators or specific reasoning algorithms is beyond the scope of this work.

Materialization and Partial Closure One can often extend the input graph with all relevant semantic consequences, pre-computed at load time, and evaluate SPARQL queries on this extended graph under the simple semantics. The approach is not applicable to entailment regimes for which one cannot pre-compute all relevant consequences, e.g., for OWL DS entailment where arbitrarily complex class expressions may be required. In the case of RDF(S) and OWL RDF-Based Semantics, however, our definitions ensure that the relevant consequences are finite and depend on the input graph only.[8]

[7] http://www.w3.org/TR/2010/WD-sparql11-entailment-20100601/

[8] Computing all such consequences for OWL RBS is of course still undecidable.

Materialization is the most common implementation technique, supported in systems such as AllegroGraph, Jena, BigOWLIM and SwiftOWLIM, Mulgara, OntoBroker, or Virtuoso.[9] The partial closure algorithm proposed in [5] for checking RDF(S) entailment can be adapted to implement the RDF(S) regime: Blank nodes in the initial graph have to marked since only they can be used in solution and instance mappings, whereas new blank nodes introduced by the partial closure algorithm cannot be used for variable bindings. Blank nodes in the query are treated as variables that are projected out immediately after BGP evaluation; the multiplicity of a solution is then given by the number of original solutions from which it can be obtained through this projection.

Query Rewriting These techniques change the query rather than the queried graph. One or more, possibly more complex queries are then evaluated over the original graph. More expressive query features may be needed, e.g., by using regular expressions to capture the transitivity of rdfs:subClassOf. To the best of our knowledge a pure query rewriting techniques has so far only be proposed for a subset of RDFS [11]. A combination with materialization, however, is also possible and successfully used, e.g., to realize RDFS entailment in Sesame [17].

Modified Query Evaluation The most direct approach for implementing our definitions is to modify existing SPARQL processors to evaluate BGPs differently. This can be accomplished, e.g., with the free ARQ library (http://jena.sourceforge.net/ARQ/). While this offers much flexibility, computing BGP matches on demand may preclude many optimizations for evaluating algebra operators. Yet, this method is a good approach for adding SPARQL support to systems that perform complex inferencing. The Hermit OWL reasoner (http://hermit-reasoner.com/) is currently being extended accordingly to support the proposed DS entailment regime. This work also includes the modification of the OWL API for parsing BGPs into extended OWL ontologies.

8 Related Work

Section 7 listed various efforts that are closely related to the implementation of our proposals. Here we focus on alternative proposals for querying expressive semantic data sources, especially for OWL.

OWL DS queries that ask for individuals and literals only are closely related to *conjunctive queries* (CQs) on description logic (DL) knowledge bases; see [4] for a basic introduction. An important difference is that CQs admit full existential variables that can represent any domain element which can be (indirectly) inferred to exist. In contrast, variables and blank nodes under OWL DS entailment may only bind to individuals that are represented by a given blank node or IRI in the input, corresponding to so-called *distinguished variables* in CQs. As of today, decidability of CQ entailment has only been established for a sublanguage of OWL 2 [13]. Restricted CQ answering still is the most common query service provided by OWL reasoners today. For example, KAON2 (http://kaon2.semanticweb.org/) and the TrOWL system (http://trowl.eu/) support the CQ subset of the OWL DS regime, whereas RacerPro (http://racer-systems.com/) has

[9] See http://en.wikipedia.org/wiki/Triplestore for more information on the mentioned systems.

its proprietary query language for CQs, called nRQL [2]. Similarly, OWLgres [16] and Quonto[10] support the CQ fragment, but they implement the OWL QL profile, which restricts the expressivity of the input ontology to allow for a more efficient implementation based on standard database techniques.

We are not aware of a complete implementation of the DS entailment regime. As of today, the Pellet OWL 2 DL reasoner (http://clarkparsia.com/pellet) is the most advanced system. The subset of SPARQL that Pellet supports – called SPARQL-DL [15] – consists of queries that can be translated into a pre-defined set of query atoms in an abstract syntax; with the semantics defined per abstract query atom.

Explicitly listing admissible queries has the advantage that one can focus on queries that are well supported by OWL reasoners. Our definition of OWL DS entailment, in contrast, uses a more general approach based on a direct mapping of BGPs to extended OWL ontologies. This allows for queries that are not typically supported by reasoners, e.g., when using variables to represent class names in complex class expressions.

Furthermore, SPARQL-DL treats blank nodes in queries like non-distinguished CQ variables with full existential meaning, whereas the DS regime treats such blank nodes like SPARQL variables that are projected out after BGP evaluation. Blank nodes under DS entailment thus are largely like distinguished CQ variables, though we allow blank nodes in the input to occur in results via Skolemization. Our design choice makes the treatment of blank nodes more uniform across all SPARQL entailment regimes, and it avoids the computational problems with non-distinguished variables in OWL.

9 Conclusions

We have presented extensions for SPARQL to incorporate RDF, RDFS, OWL RDF-Based semantics, and OWL Direct Semantics entailment. When comparing the individual entailment regimes, we find that a surprisingly high level of compatibility can be achieved between the different formalisms.

The presented regimes are closely related to the SPARQL Entailment Regimes document currently developed in the W3C SPARQL working group and we believe that our extended discussions and the resulting definitions provide a useful resource for implementers and users of SPARQL.

Our work also provides a basis for further extensions of SPARQL. Entailment regimes such as D-entailment can easily be added. A RIF entailment regime is also currently under development in the SPARQL Working Group, although some preliminaries still have to be clarified, e.g., how an RDF graph can import or encode a RIF rule set. An integration of new SPARQL operators, which are defined algebraically such as the *minus* operator currently under discussion, is straightforward. SPARQL modifications that introduce extension points besides BGP matching, in contrast, would require more considerations. Depending on the outcome of current discussions, this might be the case for *path expressions* in SPARQL 1.1. Yet, our overall impression is that SPARQL is ready – both theoretically and practically – for taking the step beyond sub-graph matching.

[10] http://www.dis.uniroma1.it/quonto/

Acknowledgements This work was supported by EPSRC in the project *HermiT: Reasoning with Large Ontologies*, and by DFG in the project *ExpresST*. We thank the members of the SPARQL working group for valuable comments and suggestions.

References

1. Beckett, D., Berners-Lee, T.: Turtle – Terse RDF Triple Language. W3C Team Submission (14 January 2008), available at http://www.w3.org/TeamSubmission/turtle/
2. Haarslev, V., Möller, R., Wessel, M.: Querying the semantic web with Racer + nRQL. In: Proc. KI-2004 International Workshop on Applications of Description Logics (2004)
3. Hayes, P. (ed.): RDF Semantics. W3C Recommendation (10 February 2004), available at http://www.w3.org/TR/rdf-mt/
4. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. Chapman & Hall/CRC (2009)
5. ter Horst, H.J.: Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. J. of Web Semantics 3(2–3), 79–115 (2005)
6. Kifer, M., Boley, H. (eds.): RIF Overview. W3C Working Group Note (22 June 2010), available at http://www.w3.org/TR/rif-overview/
7. Motik, B., Patel-Schneider, P.F., Cuenca Grau, B. (eds.): OWL 2 Web Ontology Language: Direct Semantics. W3C Recommendation (27 October 2009), available at http://www.w3.org/TR/owl2-direct-semantics/
8. Motik, B., Patel-Schneider, P.F., Parsia, B. (eds.): OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. W3C Recommendation (27 October 2009), available at http://www.w3.org/TR/owl2-syntax/
9. Patel-Schneider, P.F., Motik, B. (eds.): OWL 2 Web Ontology Language: Mapping to RDF Graphs. W3C Recommendation (27 October 2009), available at http://www.w3.org/TR/owl2-mapping-to-rdf/
10. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. ACM Transactions on Database Systems 34(3), 1–45 (2009)
11. Pérez, J., Arenas, M., Gutierrez, C.: nSPARQL: A navigational language for RDF. J. of Web Semantics (2010), to appear, http://web.ing.puc.cl/~jperez/papers/jws2010.pdf
12. Prud'hommeaux, E., Seaborne, A. (eds.): SPARQL Query Language for RDF. W3C Recommendation (15 January 2008), available at http://www.w3.org/TR/rdf-sparql-query/
13. Rudolph, S., Glimm, B.: Nominals, inverses, counting, and conjunctive queries. J. of Artificial Intelligence Research (2010), accepted for publication, http://www.comlab.ox.ac.uk/files/2175/paper.pdf
14. Schneider, M. (ed.): OWL 2 Web Ontology Language: RDF-Based Semantics. W3C Recommendation (27 October 2009), available at http://www.w3.org/TR/owl2-rdf-based-semantics/
15. Sirin, E., Parsia, B.: SPARQL-DL: SPARQL query for OWL-DL. In: Golbreich, C., Kalyanpur, A., Parsia, B. (eds.) Proc. OWLED 2007 Workshop on OWL: Experiences and Directions. CEUR Workshop Proceedings, vol. 258. CEUR-WS.org (2007)
16. Stocker, M., Smith, M.: Owlgres: A scalable OWL reasoner. In: Dolbear, C., Ruttenberg, A., Sattler, U. (eds.) Proc. OWLED 2008 Workshop on OWL: Experiences and Directions. CEUR Workshop Proceedings, vol. 432. CEUR-WS.org (2008)
17. Stuckenschmidt, H., Broekstra, J., Amerfoort, A.: Time – space trade-offs in scaling up RDF Schema reasoning. In: WISE 2005 Workshops. LNCS, vol. 3807, pp. 172–181. Springer (2005)