SPARQL Queries over Ontologies Under the Fixed-Domain Semantics

Sebastian Rudolph(✉), Lukas Schweizer(✉), and Zhihao Yao(✉)

Computational Logic Group, TU Dresden, Dresden, Germany
{sebastian.rudolph,lukas.schweizer,zhihao.yao}@tu-dresden.de

Abstract. Fixed-domain reasoning over OWL ontologies is adequate in certain closed-world scenarios and has been shown to be both useful and feasible in practice. However, the reasoning modes hitherto supported by available tools do not include querying. We provide the formal foundations of querying under the fixed domain semantics, based on the principle of certain answers, and show how fixed-domain querying can be incorporated in existing reasoning methods using answer set programming (ASP).

1 Introduction

Semantic web technologies [13] are widely adopted for knowledge representation on the Web or in other scenarios requiring intelligent data management. For expressing sophisticated background knowledge, the ontology language OWL 2 and its profiles are the standard [17,30]. OWL 2 is based on expressive description logics [4,21] and supported by optimized engines for reasoning and querying [12,28,29].

The success of OWL 2 has led to its usage also in scenarios that actually go against its standard semantics, which operates under the open-world assumption. In many such scenarios, the involved elements (the "domain") are actually known upfront. In order to better account for such scenarios, an alternative, "fixed-domain" semantics has been proposed and tools providing reasoning support have been implemented on top of answer-set solvers [9,24,25].

While the existing reasoning support is helpful for standard reasoning tasks such as satisfiability testing and also for non-standard ones such as model enumeration, sometimes more elaborate information needs must be addressed. For sophisticated querying tasks in the Semantic Web setting, SPARQL has been established as the query language of choice [31], originally designed as querying formalism for RDF graphs [27]. The recent SPARQL 1.1 standard, however, supports queries over OWL ontologies by means of the so called *entailment regimes* [5]. Given that querying OWL ontologies even under very basic queries is not known to be decidable [23], the proposed approach constitutes a compromise, implementing what is practically feasible under the open world semantics.

Under the fixed-domain semantics, however, a tighter integration of OWL background knowledge and querying can be realized without risking decidability.

A. C. Nayak and A. Sharma (Eds.): PRICAI 2019, LNAI 11670, pp. 486–499, 2019.
https://doi.org/10.1007/978-3-030-29908-8_39

Table 1. Syntax and semantics of role and concept constructors in $\mathcal{SROIQ}$, where $a_1, \ldots a_n$ denote individual names, s a role name, r a role expression and C and D concept expressions.

Name	Syntax	Semantics
Inverse role	s^-	$\{(x,y) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (y,x) \in s^{\mathcal{I}}\}$
Universal role	u	$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Top	$\top$	$\Delta^{\mathcal{I}}$
Bottom	$\bot$	$\emptyset$
Negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Nominals	$\{a_1, \ldots, a_n\}$	$\{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\}$
Univ. restriction	$\forall r.C$	$\{x \mid \forall y.(x,y) \in r^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
Exist. restriction	$\exists r.C$	$\{x \mid \exists y.(x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
Self concept	$\exists r.Self$	$\{x \mid (x,x) \in r^{\mathcal{I}}\}$
Qualified number	$\leqslant n\, r.C$	$\{x \mid \#\{y \in C^{\mathcal{I}} \mid (x,y) \in r^{\mathcal{I}}\} \leq n\}$
Restriction	$\geqslant n\, r.C$	$\{x \mid \#\{y \in C^{\mathcal{I}} \mid (x,y) \in r^{\mathcal{I}}\} \geq n\}$

Under these circumstances we can realize querying following the principle of *certain answers*: each fixed-domain model of a given ontology can be conceived as an RDF graph which can be SPARQL-queried in separation. Only if a query answer is returned when querying each and every model, it qualifies as query answer for the corresponding ontology.

Since model enumeration is a task readily provided by existing fixed-domain reasoners, the above definition immediately gives rise to a brute-force algorithm for fixed-domain ontological querying. However, the combinatorial explosion typically occurring in model-enumeration makes the feasibility of such an approach appear highly doubtful. We therefore propose an alternative method based on a tighter integration with existing reasoning technology, where SPARQL query evaluation is encoded in the same answer set program that produces the models. By means of this tight integration, we can leverage the structural similarity of certain answers and skeptical consequences.

2 Description Logics

OWL 2 DL, the version of the Web Ontology Language we focus on, is based on description logics (DLs, [4,21]). We briefly recap the description logic $\mathcal{SROIQ}$ (for details see [14]). Let N_I, N_C, and N_R be finite, disjoint sets called *individual names*, *concept names*, and *role names*, respectively.[1] These atomic entities can be used to form complex ones as displayed in Table 1.

[1] To ensure compatibility with their later usage in RDF and SPARQL, we silently presume that all these vocabulary elements are Internationalized Resource Identifiers (IRIs).

Table 2. Syntax and semantics of $\mathcal{SROIQ}$ axioms.

Axiom α	$\mathcal{I} \models \alpha$, if	
$r_1 \circ \cdots \circ r_n \sqsubseteq r$	$r_1^{\mathcal{I}} \circ \cdots \circ r_n^{\mathcal{I}} \subseteq r^{\mathcal{I}}$	RBox $\mathcal{R}$
$\mathsf{Dis}(s, r)$	$s^{\mathcal{I}} \cap r^{\mathcal{I}} = \emptyset$	
$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$	TBox $\mathcal{T}$
$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$	ABox $\mathcal{A}$
$r(a, b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$	
$a \doteq b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$	
$a \not\doteq b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$	

A $\mathcal{SROIQ}$ *knowledge base* $\mathcal{K}$ is a tuple $(\mathcal{A}, \mathcal{T}, \mathcal{R})$ where $\mathcal{A}$ is a $\mathcal{SROIQ}$ ABox, $\mathcal{T}$ is a $\mathcal{SROIQ}$ TBox and $\mathcal{R}$ is a $\mathcal{SROIQ}$ RBox. Table 2 presents the respective axiom types available in the three parts.[2] We use $N_I(\mathcal{K})$, $N_C(\mathcal{K})$, and $N_R(\mathcal{K})$ to denote the sets of individual names, concept names, and role names occurring in $\mathcal{K}$, respectively.

The semantics of $\mathcal{SROIQ}$ is defined via interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ composed of a non-empty set $\Delta^{\mathcal{I}}$ called the *domain of* $\mathcal{I}$ and a function $\cdot^{\mathcal{I}}$ mapping individual names to elements of $\Delta^{\mathcal{I}}$, concept names to subsets of $\Delta^{\mathcal{I}}$, and role names to subsets of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. This mapping is extended to complex role and concept expressions (cf. Table 1) and finally used to define satisfaction of axioms (see Table 2). We say that $\mathcal{I}$ *satisfies* a knowledge base $\mathcal{K} = (\mathcal{A}, \mathcal{T}, \mathcal{R})$ (or $\mathcal{I}$ is a *model* of $\mathcal{K}$, written: $\mathcal{I} \models \mathcal{K}$) if it satisfies all axioms of $\mathcal{A}$, $\mathcal{T}$, and $\mathcal{R}$. We say that a knowledge base $\mathcal{K}$ *entails* an axiom α (written $\mathcal{K} \models \alpha$) if all models of $\mathcal{K}$ are models of α.

Example 1. Consider a knowledge base $\mathcal{K} = (\mathcal{A}, \mathcal{T}, \mathcal{R})$. Let $\mathcal{A}$ contain the assertions $\mathtt{Aca}(\mathtt{alice})$, $\mathtt{Aca}(\mathtt{bob})$, $\mathtt{Aca}(\mathtt{claire})$, $\mathtt{Aca}(\mathtt{david})$, $\mathtt{Aca}(\mathtt{eve})$, stating that the mentioned individuals are all academics and the assertions $\mathtt{supervises}(\mathtt{alice}, \mathtt{bob})$, $\mathtt{supervises}(\mathtt{bob}, \mathtt{claire})$, and $\mathtt{supervises}(\mathtt{david}, \mathtt{eve})$ indicating supervision relationships and $\mathtt{inProject}(\mathtt{bob}, \mathtt{projectX})$, $\mathtt{inProject}(\mathtt{david}, \mathtt{projectY})$, as well as $\mathtt{inProject}(\mathtt{eve}, \mathtt{projectY})$ to indicate research project affiliations.

Let $\mathcal{T}$ contain the axioms $\mathtt{Aca} \sqsubseteq \mathtt{Masterstudent} \sqcup \mathtt{PhDstudent} \sqcup \mathtt{Professor}$ as well as $\mathtt{Masterstudent} \sqsubseteq \neg\mathtt{PhDstudent}$, $\mathtt{Masterstudent} \sqsubseteq \neg\mathtt{Professor}$, and $\mathtt{PhDstudent} \sqsubseteq \neg\mathtt{Professor}$ to indicate that every academic must be in exactly one of the three categories. Moreover, we

[2] The original definition of $\mathcal{SROIQ}$ contained more RBox axioms (expressing transitivity, (a)symmetry, (ir)reflexivity of roles), but these can be shown to be syntactic sugar. Moreover, the definition of $\mathcal{SROIQ}$ contains so-called *global restrictions* which prevents certain axioms from occurring together. These complicated restrictions, while crucial for the decidability of classical reasoning in $\mathcal{SROIQ}$ are not necessary for fixed-domain reasoning considered here, hence we omit them for the sake of brevity.

impose some constraints on supervision relationships: $\exists\texttt{supervises}.\top \sqsubseteq (\texttt{Professor} \sqcup \texttt{PhDstudent}) \sqcap \forall\texttt{supervises}.(\texttt{Masterstudent} \sqcup \texttt{PhDstudent})$ as well as $\exists\texttt{supervises}.\texttt{PhDstudent} \sqsubseteq \texttt{Professor}$ and $\texttt{PhDstudent} \sqsubseteq \forall\texttt{supervises}.\texttt{Masterstudent}$.

It can be readily checked that $\mathcal{K}$ is satisfiable. It would, however, become unsatisfiable upon adding the assertion $\texttt{supervises}(\texttt{finn}, \texttt{alice})$. Note also that, e.g., $\mathcal{K} \models \neg\texttt{Masterstudent}(\texttt{david})$.

3 Fixed-Domain Semantics

In DLs, models can be of arbitrary cardinality – for a satisfiability check, for example, all what matters is the mere existence of a model. Yet, in many applications, the domain of interest is known to be finite. Restricting reasoning to models of finite domain size (called *finite model reasoning*, a natural assumption in database theory), has been intensively studied in DLs [7,16,20,22]. As opposed to assuming the domain to be merely finite (but of arbitrary, unknown size), one can consider the case where the domain has an *a priori known cardinality* and use the term *fixed domain* [9].

Definition 1 (Fixed-Domain Semantics). *Given a non-empty finite set* $\Delta \subseteq N_I$*, called* fixed domain*, an interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *is said to be* Δ-fixed *(or just* fixed*, if* Δ *is clear from the context), if* $\Delta^{\mathcal{I}} = \Delta$ *and* $a^{\mathcal{I}} = a$ *for all* $a \in \Delta$*. Accordingly, for a DL knowledge base* $\mathcal{K}$*, we call an interpretation* $\mathcal{I}$ *a* Δ-model *of* $\mathcal{K}$*, if* $\mathcal{I}$ *is a* Δ*-fixed interpretation and* $\mathcal{I} \models \mathcal{K}$*. A knowledge base* $\mathcal{K}$ *is called* Δ-satisfiable *if it has a* Δ*-model. We say* $\mathcal{K}$ Δ-entails *an axiom* α *(*$\mathcal{K} \models_\Delta \alpha$*) if every* Δ*-model of* $\mathcal{K}$ *is also a model of* α*.*

Example 2. Consider the knowledge base $\mathcal{K}$ from Example 1. Assume, we let $\Delta = \{\texttt{alice}, \texttt{bob}, \texttt{claire}, \texttt{david}, \texttt{eve}, \texttt{projectX}, \texttt{projectY}\}$. It is not hard to see that $\mathcal{K}$ is Δ-satisfiable. Moreover, $\mathcal{K}$ Δ-entails the axiom $\neg\texttt{Aca} \sqsubseteq \{\texttt{projectX}, \texttt{projectY}\}$, whereas this axiom is not generally entailed.

4 RDF

We will now very briefly introduce RDF [8], and show how to represent a Δ-fixed interpretation as *RDF graph* which in our setting will serve as essential data structure over which SPARQL queries are evaluated. We will omit named graphs from our presentation as they are not meaningful in our context.

Let I, B, L be countably infinite, pairwise disjoint sets, called *IRIs*, *blank nodes*, and *RDF literals*, respectively. A tuple $(v_1, v_2, v_3) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an *RDF triple*, where v_1 is called the *subject*, v_2 the *predicate*, and v_3 the *object*. An *RDF graph* G (or just graph) is a set of RDF triples, and we use $\mathsf{term}(G)$ as the set of all elements from $I \cup B \cup L$ occurring in G, and $\mathsf{blank}(G) \subseteq B$ to denote the set blank nodes occurring in G. We will make later use of Definition 2 that defines the construction of an RDF graph given a Δ-fixed interpretation, promoting the interpretation as queryable artifact.

Definition 2. *Let $\mathcal{I}$ be a Δ-fixed interpretation. Then the RDF graph $G(\mathcal{I})$ induced by $\mathcal{I}$ consists of the triples $(a, \mathsf{rdf{:}type}, C)$ for all $a \in C^{\mathcal{I}}$, and (a, r, b) for all $(a, b) \in r^{\mathcal{I}}$.*

5 SPARQL

We will give a very compact introduction on the core elements of SPARQL [31], similar to [3,19]. For reasons of space and relevance, we will focus on SELECT queries and omit aggregates and solution modifiers.

Let V be a countably infinite set of available variables, where $V \cap (I \cup B \cup L) = \emptyset$. A tuple from $(I \cup L \cup V) \times (I \cup L \cup V) \times (I \cup V)$ is called *triple pattern*, and we call a finite set of triple patterns a *basic graph pattern*. Complex *graph patterns* are now inductively defined: (i) every basic graph pattern is a graph pattern, (ii) for graph patterns P_1 and P_2, the expressions P_1 AND P_2, P_1 UNION P_2, P_1 MINUS P_2, and P_1 OPT P_2 are graph patterns and (iii) for P a graph pattern and C a filter constraint (defined below), P FILTER C is a graph pattern. The set of variables occurring in a graph pattern P is denoted with $\mathsf{var}(P)$. A *filter constraint* is defined recursively as follows: (i) if $?X, ?Y \in V$ and $u \in I \cup L$ then $?X = u$, $?X = ?Y$, $\mathsf{bound}(?X)$, $\mathsf{isIRI}(?X)$, $\mathsf{isLiteral}(?X)$, and $\mathsf{isBlank}(?X)$ are *atomic filter constraints*; (ii) if C_1 and C_2 are filter constraints then $(\neg C_1)$, $(C_1 \wedge C_2)$, and $(C_1 \vee C_2)$ are *complex filter constraints*.

Finally, a SPARQL *query* q is a structure SELECT $?X_1 \ldots ?X_n$ WHERE P with $?X_1, \ldots, ?X_n$ variables and P a graph pattern. We use $\mathsf{avar}(q) = \{?X_1, \ldots, ?X_n\}$ to denote the set of *answer variables*.

Example 3. In the following, a simple SPARQL query q_1 asks for all projects in which some PhD student is involved.

```
SELECT ?Y
WHERE { ?X rdf:type PhDStudent. ?X inProject ?Y }
```

The next SPARQL query q_2 retrieves employees who are PhD students or professors together with their projects.

```
SELECT ?X ?Y
WHERE { { ?X rdf:type PhDStudent. UNION ?X rdf:type Professor. }
        AND ?X inProject ?Y. }
```

A *mapping* μ is a partial function $\mu : V \to (I \cup B \cup L)$. The domain of μ, $\mathsf{dom}(\mu) \subseteq V$, are the variables for which μ is defined. Two mappings μ_1, μ_2 are *compatible*, written $\mu_1 \sim \mu_2$, if for all $?X \in \mathsf{dom}(\mu_1) \cap \mathsf{dom}(\mu_2)$, it holds that $\mu_1(?X) = \mu_2(?X)$. Given a triple pattern t, we let $t\mu$ denote the triple obtained by replacing every variable $?X \in \mathsf{dom}(\mu)$ in t by $\mu(?X)$.

Definition 3. *Let t be a triple pattern, P, P_1, P_2 graph patterns, and G an RDF graph, then the evaluation $\langle\!\langle \cdot \rangle\!\rangle_G$ is defined as:*

$$\begin{aligned}
\langle\!\langle \{t_1, ..., t_k\} \rangle\!\rangle_G &= \{\mu \mid \mathsf{dom}(\mu) = \bigcup_{1 \leq i \leq k} \mathsf{var}(t_i) \text{ and } \{t_1\mu, ..., t_k\mu\} \subseteq G\} \\
\langle\!\langle P_1 \text{ AND } P_2 \rangle\!\rangle_G &= \{\mu_1 \cup \mu_2 \mid \mu_1 \in \langle\!\langle P_1 \rangle\!\rangle_G, \mu_2 \in \langle\!\langle P_2 \rangle\!\rangle_G, \mu_1 \sim \mu_2\} \\
\langle\!\langle P_1 \text{ UNION } P_2 \rangle\!\rangle_G &= \langle\!\langle P_1 \rangle\!\rangle_G \cup \langle\!\langle P_2 \rangle\!\rangle_G \\
\langle\!\langle P_1 \text{ MINUS } P_2 \rangle\!\rangle_G &= \langle\!\langle P_1 \rangle\!\rangle_G \setminus \langle\!\langle P_2 \rangle\!\rangle_G \\
\langle\!\langle P_1 \text{ OPT } P_2 \rangle\!\rangle_G &= \{\mu_1 \cup \mu_2 \mid \mu_1 \in \langle\!\langle P_1 \rangle\!\rangle_G, \mu_2 \in \langle\!\langle P_2 \rangle\!\rangle_G, \mu_1 \sim \mu_2\} \\
&\quad \cup \{\mu_1 \mid \mu_1 \in \langle\!\langle P_1 \rangle\!\rangle_G, \forall \mu_2 \in \langle\!\langle P_2 \rangle\!\rangle_G . \mu_1 \not\sim \mu_2\} \\
\langle\!\langle P \text{ FILTER } C \rangle\!\rangle_G &= \{\mu \in \langle\!\langle P \rangle\!\rangle_G \mid C\mu = \top\} \\
\langle\!\langle \text{SELECT } ?X_1 ... ?X_n \text{ WHERE } P \rangle\!\rangle_G &= \{\mu|_{\{?X_1, ..., ?X_n\}} \mid \mu \in \langle\!\langle P \rangle\!\rangle_G\}
\end{aligned}$$

Let C, C_1, C_2 be filter constraints, $?X, ?Y \in V$, $a \in I \cup B \cup L$. The valuation of C on a mapping μ, written $C\mu$ takes one of the three values $\{\top, \bot, \epsilon\}$ and is defined as follows. $C\mu = \epsilon$, if:

$$\begin{aligned}
&C = \mathsf{isBlank}(?X),\ C = \mathsf{isIRI}(?X),\ C = \mathsf{isLiteral}(?X), \text{ or} && (1)\\
&C = (?X = a) \text{ with } ?X \notin \mathsf{dom}(\mu); \\
&C = (?X = ?Y) \text{ with } ?X \notin \mathsf{dom}(\mu) \text{ or } ?Y \notin \mathsf{dom}(\mu); && (2)\\
&C = (\neg C_1) \text{ where } C_1\mu = \epsilon; && (3)\\
&C = (C_1 \vee C_2) \text{ with } \top \notin \{C_1\mu, C_2\mu\} \text{ and } \epsilon \in \{C_1\mu, C_2\mu\}; && (4)\\
&C = (C_1 \wedge C_2) \text{ with } \bot \notin \{C_1\mu, C_2\mu\} \text{ and } \epsilon \in \{C_1\mu, C_2\mu\}. && (5)
\end{aligned}$$

$C\mu = \top$, if:

$$\begin{aligned}
&C = \mathsf{bound}(?X) \text{ with } ?X \in \mathsf{dom}(\mu); && (1)\\
&C = \mathsf{isBlank}(?X) \text{ with } ?X \in \mathsf{dom}(\mu) \text{ and } \mu(?X) \in B; && (2)\\
&C = \mathsf{isIRI}(?X) \text{ with } ?X \in \mathsf{dom}(\mu) \text{ and } \mu(?X) \in I; && (3)\\
&C = \mathsf{isLiteral}(?X) \text{ with } ?X \in \mathsf{dom}(\mu) \text{ and } \mu(?X) \in L; && (4)\\
&C = (?X = a) \text{ with } ?X \in \mathsf{dom}(\mu) \text{ and } \mu(?X) = a; && (5)\\
&C = (?X = ?Y) \text{ with } ?X, ?Y \in \mathsf{dom}(\mu) \text{ and } \mu(?X) = \mu(?Y); && (6)\\
&C = (\neg C_1) \text{ with } C_1\mu = \bot; && (7)\\
&C = (C_1 \vee C_2) \text{ with } C_1\mu = \top \text{ or } C_2\mu = \top; && (8)\\
&C = (C_1 \wedge C_2) \text{ with } C_1\mu = \top \text{ and } C_2\mu = \top. && (9)
\end{aligned}$$

$C\mu = \bot$, otherwise.

6 SPARQL over Knowledge Bases Under Fixed Domain Semantics

In database theory, as it is the case for SPARQL, a database instance is typically conceived to be complete in terms of knowledge, and thus queries are

answered under the closed-world assumption (e.g. a person not listed in an employee database is not an employee) [2]. In contrast, a DL knowledge base represents incomplete knowledge, thus the mere absence of a fact does not allow to assume its truth value to be *false*. Alike the notion of axiom entailment, this has coined the notion of *certain query answers* [1], where (intuitively) a tuple is considered to be an answer if it is the result of evaluating the query over every model of the knowledge base. Thus, each interpretation $\mathcal{I}$ is seen as database instance, over which the query is evaluated. For the evaluation of a SPARQL query over some model $\mathcal{I}$, we will therefore use the RDF graph $G(\mathcal{I})$ induced by $\mathcal{I}$, as introduced in Sect. 4. To obtain the certain answers to a SPARQL query, we collect only those answers that are returned upon executing the query over the RDF graph $G(\mathcal{I})$ of each and every model $\mathcal{I}$ of the queried knowledge base $\mathcal{K}$.

Definition 4. *The set of* certain answers *to a SPARQL query q over a DL knowledge base $\mathcal{K}$ and a fixed domain Δ, is defined by* $\mathsf{cert}_\Delta(\mathcal{K}, q) = \{\mu \mid \mu \in \langle\!\langle q \rangle\!\rangle_{G(\mathcal{I})}$ *for all* $\mathcal{I} \models_\Delta \mathcal{K}\}$.

Example 4. Consider the knowledge base $\mathcal{K}$ from Example 1. Like in Example 2 we let $\Delta = \{\mathtt{alice}, \mathtt{bob}, \mathtt{claire}, \mathtt{david}, \mathtt{eve}, \mathtt{projectX}, \mathtt{projectY}\}$. For q_1 from Example 3 we obtain $\mathsf{cert}_\Delta(q_1, \mathcal{K}) = \{?Y \mapsto \mathtt{projectX}\}$. For q_2 we get $\mathsf{cert}_\Delta(q_2, \mathcal{K}) = \{(?X \mapsto \mathtt{bob}, ?Y \mapsto \mathtt{projectX}), (?X \mapsto \mathtt{david}, ?Y \mapsto \mathtt{projectY})\}$.

7 Practical SPARQL Answering

Practical fixed-domain reasoning for DL knowledge bases has been realized via a translation-based approach [9]. The given finite domain allows to translate DL axioms into ASP rules, and thereby make use of modern solvers to evaluate the resulting program in order to check satisfiability, as well as enumerating models – which in turn correspond to answer sets.

In consequence, it is a straightforward idea to build on top of this translation to answer SPARQL queries, in particular since translating SPARQL to datalog rules has already been proposed [3,19]; in fact, it was shown that SPARQL is equally expressive as non-recursive safe datalog with default negation.

We essentially combine both approaches (ASP-based model enumeration and ASP-based query evaluation) and adapt them to make them compatible. After providing a short introduction of answer set programming, we will sketch the translation of DL knowledge bases into answer set programs [9]. In more detail, the translation of SPARQL queries into a stratified answer set program is given thereafter.

7.1 Answer Set Programming

We review the basic notions of answer set programming [18] under the stable-model semantics [11], for further details we refer to [6,10].

We fix a countable set $\mathcal{U}$ of *(domain) elements*, also called *constants*; and presume a total order $<$ over the domain elements. An *atom* is an expression $p(t_1, \ldots, t_n)$, where p is a *predicate* of arity $n \geq 0$ and each t_i is either a variable or an element from $\mathcal{U}$. An atom is *ground* if it is free of variables. $B_{\mathcal{U}}$ denotes the set of all ground atoms over $\mathcal{U}$. A *(normal) rule* ρ is of the form

$$a \leftarrow b_1, \ldots, b_k, \ \mathsf{not}\, b_{k+1}, \ldots, \ \mathsf{not}\, b_m.$$

with $m \geq k \geq 0$, where a is an atom or empty (in the latter case the rule is called *integrity constraint*), $b_1, \ldots, b_m$ are atoms, and "not" denotes *default negation*. The *head* of ρ is the singleton set $H(\rho) = \{a\}$ if a is an atom and $H(\rho) = \emptyset$ otherwise, and the *body* of ρ is $B(\rho) = \{b_1, \ldots, b_k, \mathsf{not}\, b_{k+1}, \ldots, \mathsf{not}\, b_m\}$. Furthermore, $B^+(\rho) = \{b_1, \ldots, b_k\}$ and $B^-(\rho) = \{b_{k+1}, \ldots, b_m\}$. A rule ρ is *safe* if each variable in ρ occurs in $B^+(r)$. A rule ρ is *ground* if no variable occurs in ρ. A *fact* is a ground rule with empty body. An *(input) database* is a set of facts. A (normal) *program* is a finite set of normal rules. For a program Π and an input database D, we often write $\Pi(D)$ instead of $D \cup \Pi$. For any program Π, let U_Π be the set of all constants appearing in Π. $Gr(\Pi)$ is the set of rules $\rho\sigma$ obtained by applying, to each rule $\rho \in \Pi$, all possible substitutions σ from the variables in ρ to elements of U_Π.

An *interpretation* $I \subseteq B_{\mathcal{U}}$ *satisfies* a ground rule ρ iff $H(\rho) \cap I \neq \emptyset$ whenever $B^+(\rho) \subseteq I$, $B^-(\rho) \cap I = \emptyset$. I satisfies a ground program Π, if each $\rho \in \Pi$ is satisfied by I. A non-ground rule ρ (resp., a program Π) is satisfied by an interpretation I iff I satisfies all groundings of ρ (resp., $Gr(\Pi)$). $I \subseteq B_{\mathcal{U}}$ is an *answer set* (also called *stable model*) of Π iff it is the subset-minimal set satisfying the *Gelfond-Lifschitz reduct* $\Pi^I = \{H(\rho) \leftarrow B^+(\rho) \mid I \cap B^-(\rho) = \emptyset, \rho \in Gr(\Pi)\}$. For a program Π, we denote the set of its answer sets by $\mathcal{S}(\Pi)$.

Consequences. We rely on two notions of consequence: Given a program Π and a ground atom α, we say that Π *cautiously entails* α, written $\Pi \models_\forall \alpha$, if $\alpha \in S$ for every answer set $S \in \mathcal{S}(\Pi)$. Likewise, we say that Π *bravely entails* α, written $\Pi \models_\exists \alpha$, if there exists an answer set $S \in \mathcal{S}(\Pi)$ with $\alpha \in S$. The set of all cautious consequences of Π is denoted $\mathsf{Cn}^\forall(\Pi)$ and the set of its brave consequences $\mathsf{Cn}^\exists(\Pi)$.

7.2 Translating DL Knowledge Bases

An ASP translation of $\mathcal{SROIQ}$ knowledge bases has been proposed in [9,26]. Intuitively, given a fixed domain, one can guess an interpretation and verify modelhood with appropriate constraints (resulting from the axioms). Thus, the key idea of the translation is that every axiom is turned into an integrity constraint, and the only rules with nonempty head are so-called "guessing rules" for the extensions of every concept and role. Following this *guess and check* approach, the translation is rather direct, for example, a simple concept subsumption $A \sqsubseteq B$ becomes a constraint of the form $\leftarrow A(X), \mathsf{not}\, B(X)$; i.e. ruling out interpretations where X is an instance of A but not of B, and hence not satisfying the subsumption.

For a DL knowledge base $\mathcal{K}$ and fixed domain Δ, let $\Pi(\mathcal{K}, \Delta)$ denote the answer set program resulting from translating $\mathcal{K}$ with respect to Δ. It is shown that every answer set $S \in \mathcal{S}(\Pi(\mathcal{K}, \Delta))$, corresponds to a Δ-model of $\mathcal{K}$, and vice versa. Hence, it is possible to obtain the corresponding RDF graph $G(\mathcal{I})$ of every model via the answer sets and evaluate a SPARQL query on it. Since the translation has been implemented and is available in the tool WOLPERTINGER [25], which is able to enumerate Δ-models, SPARQL query evaluation could be realized with only little implementation effort; i.e. retrieve all models and evaluate the query on each of the induced graphs, and compute the intersection of all answers – that would be taking Definition 4 literally. However, as the sets of enumerated models tend to be very large due to combinatorial explosion, we are certain that this approach would not be feasible. Therefore, we will propose another translation-based approach.

In $\Pi(\mathcal{K}, \Delta)$, predicate names directly correspond to concept and role names in $\mathcal{K}$. This translation can syntactically be lifted to a triple notation, such that, e.g. , translating $A \sqsubseteq B$ results in the constraint $\leftarrow$ $\mathit{triple}(X, \mathsf{rdf{:}type}, A), \mathsf{not}\ \mathit{triple}(X, \mathsf{rdf{:}type}, B)$. We let $\Pi_{\mathrm{RDF}}(\mathcal{K}, \Delta)$ denote this lifted program. Now by letting $\mathrm{RDF}(S) = \{(v1, v2, v3) \mid \mathit{triple}(v1, v2, v3) \in S\}$, we obtain the following correspondence.

Lemma 1. *Let $\mathcal{K}$ be a DL knowledge base, Δ a fixed domain, and $\mathcal{I}$ a Δ-fixed interpretation. Then $\mathcal{I} \models_\Delta \mathcal{K}$ if and only if there exists some answer set $S \in \mathcal{S}(\Pi_{\mathrm{RDF}}(\mathcal{K}, \Delta))$ such that $G(\mathcal{I}) = \mathrm{RDF}(S)$.*

This correspondence now provides us with the right starting point for applying the SPARQL querying – again via a translation into ASP.

7.3 Translating SPARQL Queries

We let $\Pi(q)$ denote the answer set program resulting from the translation of a SPARQL query q, into rules, closely following [19]. Intuitively, the translation follows the recursive definition of $\langle\!\langle q \rangle\!\rangle_G$ (cf. Definition 3), evaluating the graph pattern P_q of q inside out. For a set of variables $V = \{X_1, \ldots, X_n\}$, we denote with $\overline{V} = (X_1, \ldots, X_n)$ the sequence of variables obtained relying on some lexicographic ordering. $\Pi(q)$ is then obtained with the initial call $\tau(\mathsf{avar}(q), P_q, 1)$ of the translation τ defined in the following. Thereby, the dedicated atom answer_i represents the result of evaluating the sub-graph pattern at position i in the query graph pattern seen as binary tree; thus, alike the definition of $\langle\!\langle q \rangle\!\rangle_G$ (cf. Definition 3), the translation τ traverses the binary tree. For the translation of filter expressions via the function Φ we refer the reader to [19].

$$\tau(V, \{T_1, \ldots, T_n\}, i) = \{\mathit{answer}_i(\overline{V}) \leftarrow \mathit{triple}(T_1), \ldots, \mathit{triple}(T_n)\} \quad \text{where } T_i = (v_i, v_i', v_i'') \text{ is a triple pattern.} \tag{1}$$

$$\tau(V, P_1 \,\mathsf{AND}\, P_2, i) = \{\mathit{answer}_i(\overline{V}) \leftarrow \mathit{answer}_{2i}(\overline{V'_{P_1}}), \mathit{answer}_{2i+1}(\overline{V'_{P_2}}),$$

$$
\begin{aligned}
& \mathit{join}_{|S_{P_{1,2}}|}(\overline{S'_{P_{1,2}}}, \overline{S''_{P_{1,2}}}, \overline{S_{P_{1,2}}})\} \\
& \cup \tau(\mathsf{var}(P_1), P_1, 2i) \cup \tau(\mathsf{var}(P_2), P_2, 2i+1) \cup \mathsf{Join}(|S_{P_{1,2}}|) \\
& \text{with } V'_{P_1} = \mathsf{var}(P_1)[S_{P_{1,2}} \to S'_{P_{1,2}}] \\
& \text{and } V'_{P_2} = \mathsf{var}(P_2)[S_{P_{1,2}} \to S''_{P_{1,2}}] \qquad (2)
\end{aligned}
$$

$$
\begin{aligned}
\tau(V, P_1 \,\mathsf{UNION}\, P_2, i) = \{ & \mathit{answer}_i(\overline{V[(V \backslash \mathsf{var}(P_1)) \to \mathsf{null}]}) \leftarrow \mathit{answer}_{2i}(\overline{\mathsf{var}(P_1)}), \\
& \mathit{answer}_i(\overline{V[(V \backslash \mathsf{var}(P_2)) \to \mathsf{null}]}) \leftarrow \mathit{answer}_{2i+1}(\overline{\mathsf{var}(P_2)})\} \\
& \cup \tau(\mathsf{var}(P_1), P_1, 2i) \cup \tau(\mathsf{var}(P_2), P_2, 2i+1) \qquad (3)
\end{aligned}
$$

$$
\begin{aligned}
\tau(V, P_1 \,\mathsf{MINUS}\, P_2, i) = \{ & \mathit{answer}_i(\overline{V[(V \backslash \mathsf{var}(P_1)) \to \mathsf{null}]}) \leftarrow \mathit{answer}_{2i}(\overline{\mathsf{var}(P_1)}), \\
& \qquad \mathsf{not}\ \mathit{answer}_{2i+1}(\overline{\mathsf{var}(P_1) \cap \mathsf{var}(P_2)})\} \\
& \cup \tau(\mathsf{var}(P_1), P_1, 2i) \cup \tau(\mathsf{var}(P_2), P_2, 2i+1) \qquad (4)
\end{aligned}
$$

$$
\tau(V, P_1 \,\mathsf{OPT}\, P_2, i) = \tau(V, P_1 \,\mathsf{AND}\, P_2, i) \cup \tau(V, P_1 \,\mathsf{MINUS}\, P_2, i) \qquad (5)
$$

$$
\tau(V, P \,\mathsf{FILTER}\, C, i) = \tau(\mathsf{var}(P), P, 2i) \cup \Phi(\mathit{answer}_i(\overline{V}) \leftarrow \mathit{answer}_{2i}(\overline{\mathsf{var}(P)}), C) \qquad (6)
$$

The translation of AND (joins), realized in Rule (2) requires some more explanation. First, the variables to join on are determined via $S_{P_{1,2}} = \mathsf{var}(P_1) \cap \mathsf{var}(P_2)$ (*shared variables*), and we denote with $S'_{P_{1,2}}$ and $S''_{P_{1,2}}$ the renamed copies of the shared variables $S_{P_{1,2}}$. For example, $S'_{P_{1,2}} = \{X'_1, \ldots, X'_n\}$ for $S_{P_{1,2}} = \{X_1, \ldots, X_n\}$. Thus, in Rule (2), the shared variables in answer_{2i} are replaced by their singly primed version, and the shared variables in answer_{2i} to their doubly primed version, respectively. The non-primed version is bound by $\mathit{join}_n(\ldots)$, which basically ensures that any value joins with null, for n shared variables. To implement this, we define the rule set $\mathsf{Join}(n)$ as follows:

$$
\begin{aligned}
& \mathit{join}(\mathsf{null}, \mathsf{null}, \mathsf{null}) \\
& \mathit{join}(X, X, X) \leftarrow \mathit{term}(X). \\
& \mathit{join}(X, \mathsf{null}, X) \leftarrow \mathit{term}(X). \quad \mathit{join}(\mathsf{null}, X, X) \leftarrow \mathit{term}(X). \\
& \mathit{join}_1(X'_1, X''_1, X_1) \leftarrow \mathit{join}(X'_1, X''_1, X_1) \\
& \mathit{join}_2(X'_1, X'_2, X''_1, X''_2, X_1, X_2) \leftarrow \mathit{join}_1(X'_1, X''_1, X_1), \mathit{join}(X'_2, X''_2, X_2) \\
& \mathit{join}_3(X'_1, X'_2, X'_3, X''_1, X''_2, X''_3, X_1, X_2, X_3) \leftarrow \mathit{join}_2(X'_1, X'_2, X''_1, X''_2, X_1, X_2), \mathit{join}(X'_3, X''_3, X_3) \\
& \vdots \\
& \mathit{join}_n(X'_1, \ldots, X'_m, \ldots, X_1, \ldots, X_n) \leftarrow \mathit{join}_{n-1}(X'_1, \ldots, X'_{n-1}, \ldots, X_1, \ldots, X_{n-1}), \\
& \mathit{join}(X'_n, X''_n, X_n)
\end{aligned}
$$

Example 5. The following rule is the result of applying $\tau(\{?Y\}, P_{q_1}, 1)$ for the query q_1 in Example 3.

$$
\mathit{answer}_1(Y) \leftarrow \mathit{triple}(X, \mathsf{rdf{:}type}, \mathsf{PhDStudent}), \mathit{triple}(X, \mathsf{inProject}, Y).
$$

For the query q_2 we obtain the following result for the computation of $\tau(\{?X, ?Y\}, P_{q_1}, 1)$ (omitting the rules defining the *join* predicate).

$$\begin{aligned}
answer_1(X,Y) &\leftarrow answer_2(X'), answer_3(X'',Y), join_1(X',X'',X).\\
answer_2(X) &\leftarrow answer_4(X).\\
answer_2(X) &\leftarrow answer_5(X).\\
answer_3(X,Y) &\leftarrow triple(X, \mathsf{inProject}, Y).\\
answer_4(X) &\leftarrow triple(X, \mathsf{rdf{:}type}, \mathsf{PhDStudent}).\\
answer_5(X) &\leftarrow triple(X, \mathsf{rdf{:}type}, \mathsf{Professor}).
\end{aligned}$$

Note that $\Pi(q)$ is stratified and hence can have only one answer set. Moreover, observe that the answer set of $\Pi(q)$ might contain instances of $answer_i$ with the null constant, with the intuitive meaning that the corresponding answer variables do not have a value assigned. In contrast the mapping μ is not defined to map variables onto null. Therefore, for some $V \supseteq \mathsf{dom}(\mu)$ let μ_V be the total function with domain V such that $\mu_V(?X) = \mu(?X)$ if $?X \in \mathsf{dom}(\mu)$ and $\mu_V(?X) = \mathsf{null}$ otherwise. Now, for an RDF graph G, let $\mathrm{ASP}(G)$ denote the translation of G into a database of *triple* atoms. Then we obtain the following lemma.

Lemma 2. *Let G be an RDF graph and let q be a SPARQL query with $V = \mathsf{avar}(q)$. Then $\mu \in \langle\!\langle q \rangle\!\rangle_G$ if and only if $\mu : V \to \mathsf{terms}(G)$ and $answer_1(\overline{V}\mu_V)$ is an element of the one and only answer set of $\Pi(q) \cup \mathrm{ASP}(G)$.*

7.4 Combining Model Generation and Querying

It is straightforward to reformulate Lemma 2 for models of our knowledge base.

Lemma 3. *Let $\mathcal{I}$ be a Δ-model for the DL knowledge base $\mathcal{K}$ and let q be a SPARQL query with $V = \mathsf{avar}(q)$. Then $\mu \in \langle\!\langle q \rangle\!\rangle_{G(\mathcal{I})}$ if and only if $\mu : V \to \mathsf{terms}(G)$ and $answer_1(\overline{V}\mu_V)$ is an element of the one and only answer set of $\Pi(q) \cup \mathrm{ASP}(G(\mathcal{I}))$.*

Now we are ready to "plug together" the results from Lemmas 1 and 3 to obtain the correctness result for the described translation.

Theorem 1. *For a DL knowledge base $\mathcal{K}$ over a fixed domain Δ, and a SPARQL query q with $V = \mathsf{avar}(q)$, it holds that $\mu \in \mathsf{cert}_\Delta(\mathcal{K}, q)$ if and only if $\mu : V \to \mathsf{terms}(G)$ and $answer_1(\overline{V}\mu_V) \in \mathsf{Cn}^\forall(\Pi_{\mathrm{RDF}}(\mathcal{K}, \Delta) \cup \Pi(q))$.*

Proof (Sketch). First, we observe that no predicate from $\Pi_{\mathrm{RDF}}(\mathcal{K}, \Delta)$ occurs in the head of any rule of $\Pi(q)$. Hence, by an application of the well-known splitting theorem [15], we can establish the following correspondence:

S is an answer set of $\Pi_{\mathrm{RDF}}(\mathcal{K}, \Delta) \cup \Pi(q)$ if and only if $S = S' \cup S''$ where S' is an answer set of $\Pi_{\mathrm{RDF}}(\mathcal{K}, \Delta)$ and S'' is an answer set of $\Pi(q) \cup S'$. (†)

Now consider a mapping μ such that $answer_1(\overline{V}\mu_V) \in \mathsf{Cn}^\forall(\Pi_{\mathrm{RDF}}(\mathcal{K}, \Delta) \cup \Pi(q))$. By the definition of cautious consequences, this means that $answer_1(\overline{V}\mu_V) \in S$ for every answer set S of $\Pi_{\mathrm{RDF}}(\mathcal{K}, \Delta) \cup \Pi(q)$. By (†),

this means that $answer_1(\overline{V}\mu_V) \in S''$ for the one and only answer set S'' of $\Pi(q) \cup S'$ for every answer set S' of $\Pi_{\mathrm{RDF}}(\mathcal{K}, \Delta)$. Now, using Lemmas 1 and 3 we find that this is the case exactly if for every Δ-model $\mathcal{I}$ of $\mathcal{K}$ (represented by $S' = \mathrm{ASP}(G(\mathcal{I}))$), we find that $\mu \in \langle\langle q \rangle\rangle_{G(\mathcal{I})}$. Now, by the definition of certain answers, the latter is the case exactly if $\mu \in \mathsf{cert}_\Delta(\mathcal{K}, q)$. □

8 Conclusion

In this paper, we introduced the formal underpinnings for answering SPARQL queries over OWL ontologies under the fixed domain semantics. As usual for query answering over expressive logics, we employ the principle of certain answers. We also proposed a way to realize this task by means of cautious inferencing over answer set programs, allowing to employ existing, highly optimized off-the-shelf machinery for that purpose.

As next steps in our research, we will evaluate the approach over synthetic and real-world data sets in order to verify the (albeit very plausible) assumption that our proposed approach is superior to the brute-force approach of enumerating and querying all models.

Beyond that, our initial work raises many interesting conceptual questions.

Adequacy of the Certain Answer Principle. On the one hand it is natural to ask for "guaranteed" results applying to all scenarios complying with the knowledge base. On the other hand, fixed-domain reasoning is often employed in the search for solutions to some sort of constraint satisfaction problem, and the enumerated models represent the solutions to that problem. In such a setting, one might also ask for *possible answers*, i.e., answers obtained from some model (rather than all of them). We foresee that such a setting can be captured by our approach in a straightforward way by considering brave consequences rather than cautious ones.

On yet another note, an alternative approach would be to conceive the set of models of a knowledge base as a collection of RDF graphs, stored together in an RDF dataset using named graphs. SPARQL queries could then be executed over this "super-model".

Aggregates. For space reasons, we refrained from addressing aggregates. The technically most straightforward (and readily implementable) way to define answers for queries featuring aggregates would again be to fully execute the query over each model separately and then intersect the result sets over all models. Such strategy might, however lead to unintuitive results. If we queried the knowledge base from Example 1 asking for academics and the number of projects each of them is in, we would get an empty result, since there exist models where every person is working in each of the project, where in the "standard model" one or no project would be assigned to every person. This observation suggests that under certain circumstances we might perform other operations than just intersection when accumulating certain answers.

Acknowledgments. We are grateful for the valuable feedback from the anonymous reviewers, which helped greatly to improve this work. This work has been funded by the European Research Council via the ERC Consolidator Grant No. 771779 (DeciGUT).

References

1. Abiteboul, S., Duschka, O.M.: Complexity of answering queries using materialized views. In: Proceedings of the 7th Symposium on Principles of Database Systems (PODS), pp. 254–263. ACM Press (1998)
2. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Boston (1995)
3. Angles, R., Gutierrez, C.: The expressive power of SPARQL. In: Sheth, A., et al. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 114–129. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88564-1_8
4. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook: Theory, Implementation, and Applications, 2nd edn. Cambridge University Press, Cambridge (2007)
5. Birte Glimm, C.O. (ed.): SPARQL 1.1 Entailment Regimes. W3C Working Draft, 21 March 2013. http://www.w3.org/TR/sparql11-entailment/
6. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Commun. ACM **54**(12), 92–103 (2011)
7. Calvanese, D.: Finite model reasoning in description logics. In: Proceedings of Description Logic Workshop, 1996. AAAI Technical Report, vol. WS-96-05, pp. 25–36. AAAI Press (1996)
8. Cyganiak, R., Wood, D., Lanthaler, M. (eds.): RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, 25 February 2014. http://www.w3.org/TR/rdf11-concepts/
9. Gaggl, S.A., Rudolph, S., Schweizer, L.: Fixed-domain reasoning for description logics. In: Proceedings of European Conference on AI (ECAI), 2016. Frontiers in Artificial Intelligence and Applications, vol. 285, pp. 819–827. IOS Press (2016)
10. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, San Rafael (2012)
11. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Gener. Comput. **9**(3/4), 365–386 (1991)
12. Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: HermiT: an OWL 2 reasoner. J. Autom. Reason. **53**(3), 245–269 (2014)
13. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. Chapman & Hall/CRC, Boca Raton (2009)
14. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR), pp. 57–67. AAAI Press (2006)
15. Lifschitz, V., Turner, H.: Splitting a logic program. In: Proceedings of the 11th International Conference on Logic Programming (ICLP), pp. 23–37. MIT Press (1994)
16. Lutz, C., Sattler, U., Tendera, L.: The complexity of finite model reasoning in description logics. Inf. Comput. **199**(1–2), 132–171 (2005)
17. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C. (eds.): OWL 2 Web Ontology Language: Profiles. W3C Recommendation, 27 October 2009. http://www.w3.org/TR/owl2-profiles/

18. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Ann. Math. Artif. Intell. **25**(3–4), 241–273 (1999)
19. Polleres, A., Wallner, J.P.: On the relation between SPARQL1.1 and answer set programming. J. Appl. Non-Class. Logics **23**(1–2), 159–212 (2013)
20. Rosati, R.: Finite model reasoning in *DL-Lite*. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 215–229. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68234-9_18
21. Rudolph, S.: Foundations of description logics. In: Polleres, A., et al. (eds.) Reasoning Web 2011. LNCS, vol. 6848, pp. 76–136. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23032-5_2
22. Rudolph, S.: Undecidability results for database-inspired reasoning problems in very expressive description logics. In: Proceedings of the 15th International Conference on the Principles of Knowledge Representation and Reasoning (KR), pp. 247–257. AAAI Press (2016)
23. Rudolph, S., Glimm, B.: Nominals, inverses, counting, and conjunctive queries or: why infinity is your friend!. J. Artif. Intell. Res. **39**, 429–481 (2010)
24. Rudolph, S., Schweizer, L.: Not too big, not too small... complexities of fixed-domain reasoning in first-order and description logics. In: Oliveira, E., Gama, J., Vale, Z., Lopes Cardoso, H. (eds.) EPIA 2017. LNCS (LNAI), vol. 10423, pp. 695–708. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65340-2_57
25. Rudolph, S., Schweizer, L., Tirtarasa, S.: Wolpertinger: a fixed-domain reasoner. In: Proceedings of the 16th International Semantic Web Conference (ISWC), Posters & Demonstrations. CEUR, vol. 1963. CEUR-WS.org (2017)
26. Rudolph, S., Schweizer, L., Tirtarasa, S.: Justifications for description logic knowledge bases under the fixed-domain semantics. In: Benzmüller, C., Ricca, F., Parent, X., Roman, D. (eds.) RuleML+RR 2018. LNCS, vol. 11092, pp. 185–200. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99906-7_12
27. Schreiber, G., Raimond, Y. (eds.): RDF 1.1 Primer. W3C Recommendation, 24 February 2014. http://www.w3.org/TR/rdf11-primer/
28. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: a practical OWL-DL reasoner. J. Web Semant. **5**(2), 51–53 (2007)
29. Steigmiller, A., Liebig, T., Glimm, B.: Konclude: system description. J. Web Semant. **27**, 78–85 (2014)
30. W3C OWL Working Group: OWL 2 Web Ontology Language: Document Overview. W3C Recommendation (2009). https://www.w3.org/TR/owl2-overview/
31. W3C SPARQL Working Group: SPARQL 1.1 Overview. W3C Recommendation, 21 March 2013. http://www.w3.org/TR/sparql11-overview/