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.
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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inverse role</td>
<td>(s^-)</td>
<td>({(x, y) \in \Delta^I \times \Delta^I \mid (y, x) \in s^I})</td>
</tr>
<tr>
<td>Universal role</td>
<td>(u)</td>
<td>(\Delta^I \times \Delta^I)</td>
</tr>
<tr>
<td>Top</td>
<td>(\top)</td>
<td>(\Delta^I)</td>
</tr>
<tr>
<td>Bottom</td>
<td>(\bot)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>Negation</td>
<td>(\neg C)</td>
<td>(\Delta^I \setminus C^I)</td>
</tr>
<tr>
<td>Conjunction</td>
<td>(C \cap D)</td>
<td>(C^I \cap D^I)</td>
</tr>
<tr>
<td>Disjunction</td>
<td>(C \cup D)</td>
<td>(C^I \cup D^I)</td>
</tr>
<tr>
<td>Nominals</td>
<td>({a_1, \ldots, a_n})</td>
<td>({a^I_1, \ldots, a^I_n})</td>
</tr>
<tr>
<td>Univ. restriction</td>
<td>(\forall r.C)</td>
<td>({x \mid \forall y. (x, y) \in r^I \rightarrow y \in C^I})</td>
</tr>
<tr>
<td>Exist. restriction</td>
<td>(\exists r.C)</td>
<td>({x \mid \exists y. (x, y) \in r^I \land y \in C^I})</td>
</tr>
<tr>
<td>Self concept</td>
<td>(\exists r.\text{Self})</td>
<td>({x \mid (x, x) \in r^I})</td>
</tr>
<tr>
<td>Qualified number</td>
<td>(\leq n r.C)</td>
<td>({x \mid #{y \in C^I \mid (x, y) \in r^I} \leq n})</td>
</tr>
<tr>
<td>Restriction</td>
<td>(\geq n r.C)</td>
<td>({x \mid #{y \in C^I \mid (x, y) \in r^I} \geq n})</td>
</tr>
</tbody>
</table>

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 SROIQ axioms.

<table>
<thead>
<tr>
<th>Axiom α</th>
<th>( \mathcal{I} \models \alpha ), if</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_1 \circ \cdots \circ r_n \subseteq r )</td>
<td>( r_1^\mathcal{I} \circ \cdots \circ r_n^\mathcal{I} \subseteq r^\mathcal{I} )</td>
</tr>
<tr>
<td>Dis(s, r)</td>
<td>( s^\mathcal{I} \cap r^\mathcal{I} = \emptyset )</td>
</tr>
<tr>
<td>( C \subseteq D )</td>
<td>( C^\mathcal{I} \subseteq D^\mathcal{I} )</td>
</tr>
<tr>
<td>( C(a) )</td>
<td>( a^\mathcal{I} \in C^\mathcal{I} )</td>
</tr>
<tr>
<td>( r(a, b) )</td>
<td>( (a^\mathcal{I}, b^\mathcal{I}) \in r^\mathcal{I} )</td>
</tr>
<tr>
<td>( a \doteq b )</td>
<td>( a^\mathcal{I} = b^\mathcal{I} )</td>
</tr>
<tr>
<td>( a \neq b )</td>
<td>( a^\mathcal{I} \neq b^\mathcal{I} )</td>
</tr>
</tbody>
</table>

A SROIQ knowledge base \( \mathcal{K} \) is a tuple \(( \mathcal{A}, \mathcal{T}, \mathcal{R} )\) where \( \mathcal{A} \) is a SROIQ ABox, \( \mathcal{T} \) is a SROIQ TBox and \( \mathcal{R} \) is a 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 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 \( \alpha \) (written \( \mathcal{K} \models \alpha \)) if all models of \( \mathcal{K} \) are models of \( \alpha \).

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

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

\(^2\) The original definition of 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 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 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: $\exists \text{supervises}.\top \sqsubseteq (\text{Professor} \sqcup \text{PhDstudent}) \cap \forall \text{supervises}.(\text{Masterstudent} \sqcup \text{PhDstudent})$ as well as $\exists \text{supervises}.\text{PhDstudent} \sqsubseteq \text{Professor}$ and $\forall \text{supervises}.\text{Masterstudent}$.

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

### 3 Fixed-Domain Semantics

In DLs, models can be of arbitrary cardinality – for a satisfiability check, for example, all that 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 $\Delta \subseteq \mathbb{N}_I$, called fixed domain, an interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ is said to be $\Delta$-fixed (or just fixed, if $\Delta$ 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 $\Delta$-model of $\mathcal{K}$, if $\mathcal{I}$ is a $\Delta$-fixed interpretation and $\mathcal{I} \models \mathcal{K}$. A knowledge base $\mathcal{K}$ is called $\Delta$-satisfiable if it has a $\Delta$-model. We say $\mathcal{K}$ $\Delta$-entails an axiom $\alpha$ ($\mathcal{K} \models_\Delta \alpha$) if every $\Delta$-model of $\mathcal{K}$ is also a model of $\alpha$.

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

### 4 RDF

We will now very briefly introduce RDF [8], and show how to represent a $\Delta$-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 $\text{term}(G)$ as the set of all elements from $I \cup B \cup L$ occurring in $G$, and $\text{blank}(G) \subseteq B$ 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 $\Delta$-fixed interpretation, promoting the interpretation as queryable artifact.
Definition 2. Let $\mathcal{I}$ be a $\Delta$-fixed interpretation. Then the RDF graph $G(\mathcal{I})$ induced by $\mathcal{I}$ consists of the triples $(a, \text{rdf:type}, C)$ 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 $P_1 \text{ AND } P_2$, $P_1 \text{ UNION } P_2$, $P_1 \text{ MINUS } P_2$, and $P_1 \text{ OPT } P_2$ are graph patterns and (iii) for $P$ a graph pattern and $C$ a filter constraint (defined below), $P \text{ FILTER } C$ is a graph pattern. The set of variables occurring in a graph pattern $P$ is denoted with $\text{var}(P)$. A filter constraint is defined recursively as follows: (i) if $?X, $?Y \in V$ and $u \in I \cup L$ then $?X = u, $?X = $?Y, \text{ bound}(?X), \text{ isIRI}(?X), \text{ isLiteral}(?X), \text{ and isBlank}(?X)$ are atomic filter constraints; (ii) if $C_1$ and $C_2$ are filter constraints then $(\neg C_1), (C_1 \land C_2), \text{ and } (C_1 \lor C_2)$ 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 $\text{avar}(q) = \{?X_1, \ldots, ?X_n\}$ 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 \text{ 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 \text{ rdf:type PhDStudent}. \text{ UNION } ?X \text{ rdf:type Professor}. } \text{ AND } ?X inProject $?Y. }$

A mapping $\mu$ is a partial function $\mu : V \rightarrow (I \cup B \cup L)$. The domain of $\mu$, $\text{dom}(\mu) \subseteq V$, are the variables for which $\mu$ is defined. Two mappings $\mu_1, \mu_2$ are compatible, written $\mu_1 \sim \mu_2$, if for all $?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$, 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 $?X \in \text{dom}(\mu)$ 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 $\llangle \cdot \rrangle_G$ is defined as:

$$\llangle \{t_1, \ldots, t_k\}\rrangle_G = \{\mu \mid \text{dom}(\mu) = \bigcup_{1 \leq i \leq k} \text{var}(t_i) \text{ and } \{t_1\mu, \ldots, t_k\mu\} \subseteq G\}$$

$$\llangle P_1 \text{ AND } P_2 \rrangle_G = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \llangle P_1 \rrangle_G, \mu_2 \in \llangle P_2 \rrangle_G, \mu_1 \sim \mu_2\}$$

$$\llangle P_1 \text{ UNION } P_2 \rrangle_G = \llangle P_1 \rrangle_G \cup \llangle P_2 \rrangle_G$$

$$\llangle P_1 \text{ MINUS } P_2 \rrangle_G = \llangle P_1 \rrangle_G \setminus \llangle P_2 \rrangle_G$$

$$\llangle P_1 \text{ OPT } P_2 \rrangle_G = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \llangle P_1 \rrangle_G, \mu_2 \in \llangle P_2 \rrangle_G, \mu_1 \sim \mu_2\}$$

$$\llangle P \text{ FILTER } C \rrangle_G = \{\mu \in \llangle P \rrangle_G \mid \mu \in \llangle C \rrangle_G\}$$

$$\llangle \text{SELECT } ?X_1, \ldots, ?X_n \text{ WHERE } P \rrangle_G = \{\mu|_{\{?X_1, \ldots, ?X_n\}} \mid \mu \in \llangle P \rrangle_G\}$$

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 $\mu$, written $C\mu$ takes one of the three values $\{\top, \bot, \epsilon\}$ and is defined as follows. $C\mu = \epsilon$, if:

$$C = \text{isBlank}(?X), \quad C = \text{isIRI}(?X), \quad C = \text{isLiteral}(?X), \quad \text{or} \quad (1)$$

$$C = (?X = a) \text{ with } ?X \notin \text{dom}(\mu); \quad (2)$$

$$C = (?X = ?Y) \text{ with } ?X \notin \text{dom}(\mu) \text{ or } ?Y \notin \text{dom}(\mu); \quad (3)$$

$$C = \neg C_1 \text{ where } C_1\mu = \epsilon; \quad (4)$$

$$C = (C_1 \lor C_2) \text{ with } \top \notin \{C_1\mu, C_2\mu\} \text{ and } \epsilon \in \{C_1\mu, C_2\mu\}; \quad (5)$$

$C\mu = \top$, if:

$$C = \text{bound}(?X) \text{ with } ?X \in \text{dom}(\mu); \quad (1)$$

$$C = \text{isBlank}(?X) \text{ with } ?X \in \text{dom}(\mu) \text{ and } \mu(?X) \in B; \quad (2)$$

$$C = \text{isIRI}(?X) \text{ with } ?X \in \text{dom}(\mu) \text{ and } \mu(?X) \in I; \quad (3)$$

$$C = \text{isLiteral}(?X) \text{ with } ?X \in \text{dom}(\mu) \text{ and } \mu(?X) \in L; \quad (4)$$

$$C = (?X = a) \text{ with } ?X \in \text{dom}(\mu) \text{ and } \mu(?X) = a; \quad (5)$$

$$C = (?X = ?Y) \text{ with } ?X, ?Y \in \text{dom}(\mu) \text{ and } \mu(?X) = \mu(?Y); \quad (6)$$

$$C = \neg C_1 \text{ with } C_1\mu = \bot; \quad (7)$$

$$C = (C_1 \lor C_2) \text{ with } C_1\mu = \top \text{ or } C_2\mu = \top; \quad (8)$$

$$C = (C_1 \land C_2) \text{ with } C_1\mu = \top \text{ and } C_2\mu = \top. \quad (9)$$

$C\mu = \bot$, 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 $I$ is seen as database instance, over which the query is evaluated. For the evaluation of a SPARQL query over some model $I$, we will therefore use the RDF graph $G(I)$ induced by $I$, 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 $G(I)$ of each and every model $I$ of the queried knowledge base $K$.

**Definition 4.** The set of certain answers to a SPARQL query $q$ over a DL knowledge base $K$ and a fixed domain $Δ$, is defined by $\text{cert}_Δ(Δ, q) = \{μ | μ ∈ ⟨⟨q⟩⟩_{G(I)} \text{ for all } I | Δ = K\}$.

**Example 4.** Consider the knowledge base $K$ from Example 1. Like in Example 2 we let $Δ = \{alice, bob, claire, david, eve, projectX, projectY\}$. For $q_1$ from Example 3 we obtain $\text{cert}_Δ(q_1, K) = \{?Y ← projectX\}$. For $q_2$ we get $\text{cert}_Δ(q_2, K) = \{(?X ← bob, ?Y ← projectX), (?X ← david, ?Y ← projectY)\}$.

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

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

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

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

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

\textbf{7.2 Translating DL Knowledge Bases}

An ASP translation of 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 \textquote{guessing rules} for the extensions of every concept and role. Following this \textit{guess and check} approach, the translation is rather direct, for example, a simple concept subsumption $A \sqsubseteq B$ becomes a constraint of the form $\neg A(X), \text{not } B(X)$; 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 $\Delta$, let $\Pi(\mathcal{K}, \Delta)$ denote the answer set program resulting from translating $\mathcal{K}$ with respect to $\Delta$. It is shown that every answer set $S \in \mathcal{S}(\Pi(\mathcal{K}, \Delta))$, corresponds to a $\Delta$-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 $\Delta$-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 $\leftarrow$ triple$(X, \text{rdf:} \text{type}, A)$, not triple$(X, \text{rdf:} \text{type}, B)$. We let $\Pi_{\text{RDF}}(\mathcal{K}, \Delta)$ denote this lifted program. Now by letting $\text{RDF}(S) = \{(v_1, v_2, v_3) \mid \text{triple}(v_1, v_2, v_3) \in S\}$, we obtain the following correspondence.

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

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 $\tau(\text{avar}(q), P_q, 1)$ of the translation $\tau$ defined in the following. Thereby, the dedicated atom $\text{answer}_i$ 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 $\tau$ traverses the binary tree. For the translation of filter expressions via the function $\Phi$ we refer the reader to [19].

\[
\tau(V, \{T_1, \ldots, T_n\}, i) = \{\text{answer}_i(\overline{V}) \leftarrow \text{triple}(T_1), \ldots, \text{triple}(T_n)\}
\]

where $T_i = (v_i, v'_i, v''_i)$ is a triple pattern. (1)

\[
\tau(V, P_1 \text{ AND } P_2, i) = \{\text{answer}_i(\overline{V}) \leftarrow \text{answer}_{2i}(\overline{V}_{P_1}), \text{answer}_{2i+1}(\overline{V}_{P_2})\},
\]

\[
\tau(V, P, i) = \{\text{answer}_i(\overline{V}) \leftarrow \text{triple}(P)\}
\]
Example 5. The following rule is the result of applying \( \tau(\{?Y\}, P_{q_1}, 1) \) for the query \( q_1 \) in Example 3.

\[
answer_1(Y) \leftarrow triple(X, rdf:type, PhDStudent), triple(X, inProject, Y).
\]
For the query $q_2$ we obtain the following result for the computation of $\tau(\{X,Y\}, P_1, 1)$ (omitting the rules defining the join predicate).

$$\begin{align*}
\text{answer}_1(X, Y) & \leftarrow \text{answer}_2(X'), \text{answer}_3(X'', Y), \text{join}_1(X', X'', X). \\
\text{answer}_2(X) & \leftarrow \text{answer}_4(X). \\
\text{answer}_2(X) & \leftarrow \text{answer}_5(X). \\
\text{answer}_3(X, Y) & \leftarrow \text{triple}(X, \text{inProject}, Y). \\
\text{answer}_4(X) & \leftarrow \text{triple}(X, \text{rdf:type}, \text{PhDStudent}). \\
\text{answer}_5(X) & \leftarrow \text{triple}(X, \text{rdf:type}, \text{Professor}).
\end{align*}$$

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 $\text{answer}_i$ with the null constant, with the intuitive meaning that the corresponding answer variables do not have a value assigned. In contrast the mapping $\mu$ is not defined to map variables onto null. Therefore, for some $V \supseteq \text{dom}(\mu)$ let $\mu_V$ be the total function with domain $V$ such that $\mu_V(?X) = \mu(?X)$ if $?X \in \text{dom}(\mu)$ and $\mu_V(?X) = \text{null}$ otherwise. Now, for an RDF graph $G$, let $\text{ASP}(G)$ 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 $V = \text{avar}(q)$. Then $\mu \in \langle \langle q \rangle \rangle_G$ if and only if $\mu : V \rightarrow \text{terms}(G)$ and $\text{answer}_1(\overline{V}\mu_V)$ is an element of the one and only answer set of $\Pi(q) \cup \text{ASP}(G)$.

### 7.4 Combining Model Generation and Querying

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

**Lemma 3.** Let $I$ be a $\Delta$-model for the DL knowledge base $\mathcal{K}$ and let $q$ be a SPARQL query with $V = \text{avar}(q)$. Then $\mu \in \langle \langle q \rangle \rangle_{G(I)}$ if and only if $\mu : V \rightarrow \text{terms}(G)$ and $\text{answer}_1(\overline{V}\mu_V)$ is an element of the one and only answer set of $\Pi(q) \cup \text{ASP}(G(I))$.

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 $\Delta$, and a SPARQL query $q$ with $V = \text{avar}(q)$, it holds that $\mu \in \text{cert}_\Delta(\mathcal{K}, q)$ if and only if $\mu : V \rightarrow \text{terms}(G)$ and $\text{answer}_1(\overline{V}\mu_V) \in \text{Cn}^\mathcal{V}(\Pi_{\text{RDF}}(\mathcal{K}, \Delta) \cup \Pi(q))$.

**Proof (Sketch).** First, we observe that no predicate from $\Pi_{\text{RDF}}(\mathcal{K}, \Delta)$ 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 $\Pi_{\text{RDF}}(\mathcal{K}, \Delta) \cup \Pi(q)$ if and only if $S = S' \cup S''$ where $S'$ is an answer set of $\Pi_{\text{RDF}}(\mathcal{K}, \Delta)$ and $S''$ is an answer set of $\Pi(q) \cup S'$.

Now consider a mapping $\mu$ such that $\text{answer}_1(\overline{V}\mu_V) \in \text{Cn}^\mathcal{V}(\Pi_{\text{RDF}}(\mathcal{K}, \Delta) \cup \Pi(q))$. By the definition of cautious consequences, this means that $\text{answer}_1(\overline{V}\mu_V) \in S$ for every answer set $S$ of $\Pi_{\text{RDF}}(\mathcal{K}, \Delta) \cup \Pi(q)$. By (†),
this means that $answer_1(\forall \mu \forall V) \in S''$ for the one and only answer set $S''$ of $\Pi(q) \cup S'$ for every answer set $S'$ of $\Pi_{\text{RDF}}(K, \Delta)$. Now, using Lemmas 1 and 3 we find that this is the case exactly if for every $\Delta$-model $I$ of $K$ (represented by $S' = \text{ASP}(G(I))$), we find that $\mu \in \langle q \rangle_{G(I)}$. Now, by the definition of certain answers, the latter is the case exactly if $\mu \in \text{cert}_{\Delta}(K, q)$. □

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