A Prolog-based Query Language for OWL

Jesús M. Almendros-Jiménez

Dpto. de Lenguajes y Computación, Universidad de Almería, Spain

Abstract

In this paper we investigate how to use logic programming (in particular, Prolog) as query language against OWL resources. Our query language will be able to retrieve data and meta-data about a given OWL based ontology. With this aim, firstly, we study how to define a query language based on a fragment of Description Logic, then we show how to encode the defined query language into Prolog by means of logic rules and finally, we identify Prolog goals which correspond to queries.

Keywords: OWL, Logic Programming, Semantic Web

1 Introduction

The Semantic Web framework [BLHL+01,EIKP08] proposes that Web data represented by HTML and XML have to be enriched by means of meta-data, in which modeling is mainly achieved by means of the Resource Description Framework (RDF) [W3C04c] and the Web Ontology Language (OWL) [W3C04a]. RDF and OWL are proposals of the W3C consortium for ontology modeling. OWL is syntactically layered on RDF whose underlying model is based on triples. The RDF Schema (RDFS) [W3C04b] is also a W3C proposal and enriches RDF with specific vocabularies for meta-data. RDFS/RDF and OWL can be used for expressing both data and meta-data. OWL can be considered as an extension of RDFS in which a richer vocabulary allows to express new relationships. OWL offers more complex relationships than RDF(S).

1 The author work has been partially supported by the Spanish MICINN under grant TIN2008-06622-C03-03.
2 Email: jalmen@ual.es
3 http://www.w3.org.

Preprint submitted to Electronic Notes in Theoretical Computer Science 9 December 2010
between entities including means to limit the properties of classes with respect to the number and type, means to infer that items with various properties are members of a particular class and a well-defined model of property inheritance. OWL is based on the so-called Description Logic (DL) [BCM+03], which is a family of logics (i.e. fragments) with different expressivity power. Most of fragments of Description Logic are subsets or variants of $C^2$, the subset of first-order logic (FOL) extended with counting quantifiers, with formulas without function symbols and maximum two variables, which is known to be decidable [GKV97]. Description Logic can therefore also be understood as an attempt to address the major drawbacks of using FOL for knowledge representation and inference, and also the syntax of DL allows a variable-free notation. The most prominent fragment of DL is $SROIQ$ which is the basis of the new standardized OWL 2. OWL 2 semantics has been defined in [W3C09a], in which a direct semantics is defined based on Description Logic, and in [W3C09b] a RDF-based semantics is provided.

In this paper we investigate how to use logic programming (in particular, Prolog) as query language against OWL resources. Our query language will be able to retrieve data and meta-data about a given OWL based ontology. With this aim, firstly, we study how to define a query language based on a fragment of Description Logic, then we show how to encode the defined query language into Prolog by means of logic rules and finally, we identify Prolog goals which correspond to queries.

Basically, our work goes towards the use of logic programming as query language for the Semantic Web. It follows our previous research line about the use of logic programming for the handling of Web data. In particular, we have studied the encoding in logic programming of the XML query language XPath in [ABE08, ABE06], and the encoding in logic programming of the XML query language XQuery in [ABE09, Alm09a], studying extensions of XQuery for the handling of RDF and OWL in [Alm08, Alm09c, Alm09b]. In this framework, we would like to study how OWL querying and reasoning can be achieved by means of logic programming in order to be integrated with the proposal of the implementation of XQuery in logic programming.

1.1 Description Logic

Description Logic is a formalism for expressing relationships between concepts, between roles, between concepts and roles, and between concepts, roles and individuals. Formulas of Description Logic can be used for representing knowledge, that is, concept descriptions, about a domain of interest. Typically, Description Logic is used for representing a TBox (terminological box) and the ABox (assertional box). The TBox describes concept (and role) hi-
erarchies (i.e., relations between concepts and roles) while the ABox contains relations between individuals, concepts and roles. Therefore we can see the TBox as the meta-data description, and the ABox as the description about data.

In this context, we can distinguish between (1) reasoning tasks and (2) querying tasks from a given ontology. In both cases, a certain inference procedure should be present in order to deduce new relationships from a given ontology. The most typical (1) reasoning tasks, with regard to a given ontology, include: (a) Instance checking, that is, whether a particular individual is a member of a given concept, (b) Relation checking, that is, whether two individuals hold a given role, (c) Subsumption, that is, whether a concept is a subset of another concept, (d) Concept consistency, that is, consistency of the concept relationships, and (e) A more general case of consistency checking is ontology consistency in which the problem is to decide whether a given ontology has a model. However, one can be also interested in (2) querying tasks such as: (a) Instance retrieval, which means to retrieve all the individuals of a given concept entailed by the ontology, and (b) Property fillers retrieval which means to retrieve all the individuals which are related to a given individual with respect to a given role.

OWL/DL reasoning is a topic of research of increasing interest in the literature. Most of DL reasoners (for instance, Racer [HMW08], FaCT++ [TH06], Pellet [SPG07]) are based on tableaux based decision procedures. Based on logic programming, the DLog system [LSK08] reasons with an ontology by means of the encoding into Prolog and the use of the PTTP theorem prover. The KAON2 tool [BEH02,HMS08] is based on the encoding of the SHIQ fragment into disjunctive Datalog [HMS07]. They have studied a resolution-based inference.

We believe that an interesting research line would be to design a query language whose aim is to express such reasoning and querying tasks. In other words, the study of some kind of formalism in which we can express the kind of task (i.e. reasoning and querying task) one want to achieve with respect to a given ontology. Such a language should be equipped with some kind of formulas for representing the query. In addition, such a query language should be equipped with a inference mechanism in order to reason with the ontology. Such inference mechanism could be based on an entailment relationship. The query language we propose will be based on Description Logic formulas which can contain free variables. Free variables represent the elements of the formula for which we want to retrieve values. Such values should represent the names of concepts, roles and individuals satisfying a given query (i.e. the DL formula). In such a case, we would obtain the answers to a given querying task. In the
case the set of answers is empty the formula is false. In the case of formulas without free variables, the query would represent a reasoning task, and the answer would be true or false.

The definition of a set of entailment rules for RDFS and OWL has attracted the attention of the Semantic Web community. An entailment relationship defines which relationships can be deduced from a given ontology. In this context, the authors of [MPG07] have observed that the rules of entailment of the official RDF Semantics specification are not complete, and have suggested for the case of RDFS, to identify a fragment which encompasses the essential features of RDFS, which preserves the original semantics, be easy to formalize and can serve to prove results about its properties. With this aim they have defined a fragment of RDFS that covers the crucial vocabulary of RDFS, they have proved that it preserves the original RDF semantics, and avoids vocabulary and axiomatic information that only serves to reason about the structure of the language itself and not about the data it describes. The studied fragment of RDFS lifts the structural information into the semantics of the language hiding them from developers and users. They have given a sound and entailment relationship for a fragment of RDF including rdf:type, rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain and rdfs:range. In the case of OWL, there is a proposal for extending rules for entailment with RDFS to OWL. The so-called pD* approach [tH04] is a proposal for an extension of the RDFS vocabulary with some elements of OWL: owl:FunctionalProperty, owl:InverseFunctionalProperty, owl:sameAs, owl:SymmetricProperty, owl:TransitiveProperty and owl:inverseOf. For such a fragment, they have defined a complete set of simple entailment rules. The pD* approach has been successfully applied to the SAOR (Scalable Authoritative OWL reasoner) [HHP08], a system which focuses on a good performance with RDF and OWL data.

In this line, our approach considers a set of entailment rules for a fragment of OWL. Such fragment differs from the fragment of the pD* [tH04], neither pD* is include in our fragment, nor our fragment is included in pD*, but our fragment includes the RDFS fragment of [MPG07]. In addition, our fragment of DL allows to encode the entailment relationship by means of logic programming, in particular, by Prolog. For this reason, we have studied the relationship between Description Logic and Logic Programming.

1.2 Description Logic and Logic Programming

In this area of research, some authors [GHVD03,Vol04] have studied the intersection of Description Logic and Logic Programming, in other words, which fragment of Description Logic can be expressed in Logic Programming. In
[GHVD03], they have defined the so-called Description Logic Programming (DLP), which corresponds with a fragment of SHIQ. With this aim, firstly, the fragment of DL is encoded into a fragment of FOL; and after the fragment of FOL is encoded into logic programming. Other fragments of OWL/DL can be also encoded into logic programming, in particular, Volz [Vol04] has encoded fragments of SHOIN into Datalog, Datalog(=), Datalog(=,IC) and Prolog(=,IC); where “=” means “with equality”, and “IC” means “with Integrity constraints”. Some other proposals have encoded the fragment SHIQ into disjunctive Datalog [HMS07], and into Datalog(IC,≠,not) [dBLPF05], where “not” means “with negation”.

In our proposal, we have focused on one of the Volz’s fragments which can be encoded into Datalog. It is a fragment of SHOIN, which includes the elements of the OWL vocabulary: rdf : type, rdfs : subClassOf, rdfs : subPropertyOf, owl : equivalentProperty, owl : equivalentClass, rdfs : domain, rdfs : range, owl : someValuesFrom, owl : has Value and owl : allValuesFrom, and handles owl : union and owl : intersection operators, and owl : TransitiveProperty, owl : SymmetricProperty and owl : inverseOf properties. All of them are used with some restrictions. We believe that our proposal could be extended to other fragments of SHOIN studied in [Vol04].

Our work can be intended as an extension of the DLP framework in the following sense. The encoding of DL into logic programming defines an entailment relationship based on the rules. However, our encoding differs from DLP encoding: instead of encoding class and role names as Prolog predicates, we encode them as Prolog atoms. In fact, our encoding only uses a predicate called triple which defines by means of Prolog facts the relationships (i.e. the ABox and TBox elements) of the ontology. Therefore the ontology can be easily stored in secondary memory with efficient access. As a consequence of such encoding concept and role names are now first-class citizens, and they can be handled as individuals in the corresponding Prolog program and goals. In addition, in our approach, and in contrast with the DLP approach, complex concepts in Description Logic are handled by means of Prolog terms. For instance, ∀P.C is represented as a Prolog term forall(P,C). It also allows to handle complex concepts also as first-class citizens. Our work can be also considered as an extension of DLP framework because we investigate a more flexible query language than in DLP. By studying the entailment relationship between Description Logic formulas of the given fragment, we are able to provide semantics to a more flexible query language than the underlying query language in the DLP framework. For instance, we are able to entail in our framework typing formulas of the style ∃P.C(I), that is, I is an individual of type ∃P.C, in the spite of ∃P.C(I) is not explicitely declared in the ABox.
But more interesting, such a typing (i.e., $\exists P.C$) can be retrieved as an answer of our query language. It is not possible with the DLP encoding because $\exists P.C$ formulas can not be handled as first-class citizens. Moreover, $P$ and $C$ can work as query result, providing more flexible queries. In other words, our DLP extension can be considered as second order extension (i.e. predicate names in DLP can be replaced by variables) of the DLP approach. For such extension, we provide an entailment relationship. The entailment relationship defined in our proposal is also an extension of the underlying entailment relationship of the DLP fragment. Our entailment relationship is able to infer new DL formulas of the kind $\exists P.C \subseteq D$ when they are not explicitely declared in the TBox. Such new DL formulas would correspond with the inference of new rules from the DLP encoding of the selected fragment. In summary, our entailment relationship can obtain the same statements about the ABox: individual assertions and property fillers than the DLP approach for the same fragment, but we can also obtain new statements about the TBox: subclass and subproperty assertions with respect to the DLP approach.

Recently, the W3C has proposed a set of entailment rules for OWL, the so-called OWL RL [W3C09c]. Such set of rules subsumes our proposed entailment relationship. Therefore, our approach can be seen as a contribution to the OWL RL framework, providing an implementation by means of Prolog rules. We have developed a prototype of our approach which can be downloaded from http://indalog.ual.es/OProlog. We have tested our prototype with several examples of ontologies including the running example presented bellow. With respect to the implementation we have to make the following remarks:

- We have implemented the OWL-based query language using the SWI-Prolog interpreter.
- We have used the RDFS library of SWI-Prolog [WSW03] for implementing loading of OWL triples. The RDFS library of SWI-Prolog has limited querying capabilities. Our proposal can be considered as an extension of such library for OWL querying.
- The syntactic structure of entailment rules makes that the implementation in Prolog loops: the predicate triple can call to itself with the same arguments. However, we have solved that problem by implementing an small Prolog interpreter which runs Prolog rules in such a way that it memorizes the facts about the predicate triple, and avoids calls with the same arguments.
- The efficiency of the prototype has been improved by rewriting some of the rules presented here.

The structure of the paper is as follows. Section 2 will present the fragment
of DL of our proposal. Section 3 will define the query language proposed from the fragment. Section 4 will describe the encoding of the query language in Prolog. Finally, Section 5 will conclude and present future work.

2 Web Ontology Language

In this section we will show what kind of ontologies will be allowed in our framework. It will also define the entailment relationship.

An ontology $\mathcal{O}$ in our framework contains a **TBox** including a sequence of definitions $T_1, \ldots, T_n$ of the form:
where \( E, F \) are class (\textit{i.e. concept}) descriptions of equivalence type (denoted by \( E, F \in \mathcal{E} \)) of the form:

\[
E ::= C_0 | E_1 \cap E_2 | \exists P.\{O\}
\]

In addition, \( C \) is a class (\textit{i.e. concept}) description of left-hand side type (denoted by \( C \in \mathcal{L} \)), of the form:

\[
C ::= C_0 | C_1 \cap C_2 | \exists P.\{O\} | C_1 \cup C_2 | \exists P.\{O\}
\]

and \( D \) is a class (\textit{i.e. concept}) description of right-hand side type (denoted by \( D \in \mathcal{R} \)), of the form:

\[
D ::= A | D_1 \cap D_2 | \exists P.\{O\} | \forall P.\{O\}
\]

In all previous cases, \( C_0 \) is an atomic class (\textit{i.e. class name}), \( P, Q \) are property (\textit{i.e. role}) names and \( O \) is an individual name. In addition, the \textbf{ABox} contains a sequence of definitions \( A_1, \ldots, A_m \) of the form:

\[
A ::= P(A,B) | C_0(A) | \top(A)
\]

where \( P \) is a property name, \( C_0 \) is a class name, and \( A, B \) are individual names.

Basically, the proposed subset of DL restricts the form of class descriptions in right and left hand sides of subclass and class equivalence definitions, and in individual assertions. Such restriction is required according to [Vol04] in order to be able to encode the fragment of DL into logic programming. Following [Vol04], the universal quantification is only allowed in the right hand sides of DL formulas, which corresponds in the encoding to the occurrence of the same quantifier in the left hand sides (\textit{i.e. heads}) of rules. The existential quantification only occurs in the right hand by the same reason. Union formulas are required to occur in the left hand sides of DL formulas, which corresponds in the encoding to the definition of two rules.

Let us see an example of an ontology \( O_0 \) (see Figure 1). The ontology \( O_0 \) describes meta-data in the \textbf{TBox} defining that the elements of \textit{Man} and the elements of \textit{Woman} are elements of \textit{Person} (cases (1) and (2)); and the elements of \textit{Paper} and elements of \textit{Book} are elements of \textit{Manuscript} (case (4)).
Table 1
Inference Calculus $\vdash_{OI}$

<table>
<thead>
<tr>
<th>Rule Name</th>
<th>Inference</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Eq1)</td>
<td>$E \equiv F \vdash_{OI} E \equiv F$</td>
</tr>
<tr>
<td>(Eq2)</td>
<td>$E \equiv F, F \equiv G \vdash_{OI} E \equiv G$</td>
</tr>
<tr>
<td>(Eq3)</td>
<td>$E \equiv F \vdash_{OI} F \equiv E$</td>
</tr>
<tr>
<td>(Eq4)</td>
<td>$E \subseteq F, F \subseteq E \vdash_{OI} E \subseteq F$</td>
</tr>
<tr>
<td>(Sub1)</td>
<td>$E \equiv F \vdash_{OI} E \subseteq F$</td>
</tr>
<tr>
<td>(Sub2)</td>
<td>$C \subseteq D, D \subseteq E \vdash_{OI} C \subseteq E$</td>
</tr>
<tr>
<td>(Sub3)</td>
<td>$C \cup D \subseteq E \vdash_{OI} C \subseteq E$</td>
</tr>
<tr>
<td>(Sub4)</td>
<td>$E \subseteq C \cap D \vdash_{OI} E \subseteq C$</td>
</tr>
<tr>
<td>(Sub5)</td>
<td>$C_1 \cap C_2 \subseteq D, E \subseteq C_1 \vdash_{OI} E \cap C_2 \subseteq D$</td>
</tr>
<tr>
<td>(Sub6)</td>
<td>$C_1 \cup C_2 \subseteq D, E \subseteq C_1 \vdash_{OI} E \cup C_2 \subseteq D$</td>
</tr>
<tr>
<td>(Sub7)</td>
<td>$C \subseteq D_1 \cap D_2, D_1 \subseteq E \vdash_{OI} C \subseteq E \cap D_2$</td>
</tr>
<tr>
<td>(Sub8)</td>
<td>$\exists P{O} \subseteq D, Q \subseteq P \vdash_{OI} \exists Q{O} \subseteq D$</td>
</tr>
<tr>
<td>(Sub9)</td>
<td>$\exists P \subseteq D, Q \subseteq P \vdash_{OI} \exists Q \subseteq D$</td>
</tr>
<tr>
<td>(Sub10)</td>
<td>$\exists P \subseteq D, E \subseteq C \vdash_{OI} \exists P \subseteq C \subseteq D$</td>
</tr>
<tr>
<td>(Sub11)</td>
<td>$C \subseteq \exists P{O}, P \subseteq Q \vdash_{OI} C \subseteq \exists Q{O}$</td>
</tr>
<tr>
<td>(Sub12)</td>
<td>$C \subseteq \forall P.D, Q \subseteq P \vdash_{OI} C \subseteq \forall Q.D$</td>
</tr>
<tr>
<td>(Sub13)</td>
<td>$C \subseteq \forall P.D, D \subseteq E \vdash_{OI} C \subseteq \forall P.E$</td>
</tr>
<tr>
<td>(Type1)</td>
<td>$C(A), C \subseteq D \vdash_{OI} D(A)$</td>
</tr>
<tr>
<td>(Type2)</td>
<td>$P(A.O) \subseteq D \vdash_{OI} D(A)$</td>
</tr>
<tr>
<td>(Type3)</td>
<td>$P(A, B), C(B) \subseteq P \subseteq C \subseteq D \vdash_{OI} D(A)$</td>
</tr>
<tr>
<td>(Type4)</td>
<td>$C(A), P(A, B), C \subseteq \forall P.D \vdash_{OI} D(B)$</td>
</tr>
<tr>
<td>(Type5)</td>
<td>$C(A), P(B, A), C \subseteq \forall P^\perp.D \vdash_{OI} D(B)$</td>
</tr>
<tr>
<td>(Type6)</td>
<td>$\text{Cond}(E_i), \bigwedge_{1 \leq i \leq n} E_i \subseteq D \vdash_{OI} D(A)$</td>
</tr>
<tr>
<td>(Type7)</td>
<td>$\vdash_{OI} \top(A)$</td>
</tr>
<tr>
<td>(Prop1)</td>
<td>$P \equiv Q, Q \equiv R \vdash_{OI} P \equiv R$</td>
</tr>
<tr>
<td>(Prop2)</td>
<td>$P \equiv Q \vdash_{OI} P \equiv Q$</td>
</tr>
<tr>
<td>(Prop3)</td>
<td>$P \equiv Q \vdash_{OI} Q \equiv P$</td>
</tr>
<tr>
<td>(Prop4)</td>
<td>$P \subseteq Q, Q \subseteq P \vdash_{OI} P \subseteq Q$</td>
</tr>
<tr>
<td>(Prop5)</td>
<td>$P \subseteq Q \vdash_{OI} Q \subseteq P$</td>
</tr>
<tr>
<td>(Prop6)</td>
<td>$P \subseteq Q, Q \subseteq R \vdash_{OI} P \subseteq R$</td>
</tr>
<tr>
<td>(Prop7)</td>
<td>$P \subseteq Q, P(A, B) \vdash_{OI} Q(A, B)$</td>
</tr>
<tr>
<td>(Prop8)</td>
<td>$P \subseteq Q^\perp, P(A, B) \vdash_{OI} Q(B, A)$</td>
</tr>
<tr>
<td>(Prop9)</td>
<td>$Q^\perp \subseteq P, Q(A, B) \vdash_{OI} P(B, A)$</td>
</tr>
<tr>
<td>(Prop10)</td>
<td>$P^\perp \subseteq P, P(A, B), P(B, C) \vdash_{OI} P(A, C)$</td>
</tr>
<tr>
<td>(Prop11)</td>
<td>$C(A), C \subseteq \exists P{O} \vdash_{OI} P(A, O)$</td>
</tr>
</tbody>
</table>
In addition, a Writer is a Person who is the author_of a Manuscript (case (3)), and the class Reviewed contains the elements of Manuscript reviewed_by a Person (case (6)). Moreover, the XMLBook class contains the elements of Manuscript which have as topic the value “XML” (case (5)). The classes Score and Topic contain, respectively, the values of the properties rating and topic associated to Manuscript (cases (7) and (8)). The property average_rating is a subproperty of rating (case (10)). The property writes is equivalent to author_of (case (9)), and authored_by is the inverse property of author_of (case (11)). Finally, the property author_of, and conversely, reviewed_by, has as domain a Person and as range a Manuscript (cases (12)-(15)).

The ABox describes data about two elements of Book: “Data on the Web” and “XML in Scotland” and a Paper: “Growing XQuery”. It describes the author_of and authored_by relationships for the elements of Book and the writes relation for the elements of Paper. In addition, the elements of Book and Paper have been reviewed and rated, and they are described by means of a topic.

2.1 Entailment Relationship

Now, we would like to show an inference calculus, denoted by ⊢_{OI}, which defines the entailment relationship between formulas of the selected fragment. The inference calculus is shown in Table 1. In the rule (Eq1) E is a class name, and D is a class name in rules (Type1) to (Type5). In the rule (Type6) A is an individual name, and in the rule (Prop1) P is a property name. In addition, in the rule (Type6) we have that Cond(E) = E(A) if E is atomic; Cond(E) = P(A, O) if E = ∃P.Q{O}, Cond(E) = P(A, B), C(B) if E = ∀P.C. Finally, in the rules (Prop1) to (Prop6), and (Sub12), (Sub13), P and Q can have the form S, S^− and S^+. The rules from (Eq1) to (Eq4) handle inference about equivalence. (Eq1) infers equivalence by reflexivity, (Eq2) infers equivalence by transitivity, and (Eq3) infers equivalence by symmetry. (Eq4) infers equivalence from the subclass relationship. The rules from (Sub1) to (Sub13) handle inference about subclasses. Cases from (Sub3) to (Sub7) define new subclass relationships from union and intersection operators. However, the calculus does not introduce new union and intersection operators. For instance C ∩ E ⊆ D is not entailed from C ⊆ D. The same can be said for C ∪ E ⊆ D which is not entailed from C ⊆ D and E ⊆ D. The same happens with C ⊆ D ∩ E from C ⊆ D and C ⊆ E. Cases from (Sub8) to (Sub13) define new subclass relationships for complex formulas. Such entailment of complex formulas is the main contribution of our inference calculus with respect to DLP framework and the pD^* approach. In the former case, the entailment relationship can only entail formulas about indi-
individual and property fillers assertions, and not about complex formulas. In the later case, $pD^*$ does not handle $\text{owl:someValuesFrom}$, $\text{owl:allValuesFrom}$ and $\text{owl:hasValue}$ and therefore such rules have not sense in $pD^*$. However, the $pD^*$ handles for instance $\text{owl:FunctionalProperty}$ which is not consider in our framework.

The rules (Type1) to (Type7) infer type relationships from subclass and equivalence relationships. The most relevant ones are the cases from (Type2) to (Type5) defining the meaning of complex formulas w.r.t. individuals. Finally, the rules (Prop1) to (Prop11) infer relationships about roles. The most relevant ones are the case (Prop8) and (Prop9) about the inverse of a property and the case (Prop10) about the transitivity relationship. Our inference calculus is able to infer new information from a given ontology. For instance, $O_0 \vdash_{O_1} \text{Reviewed("Data on the Web")}$, using the following TBox and ABox information of $O_0$:

\[
\begin{align*}
\text{Book("Data on the Web").} \\
\text{Book} \sqsubseteq \text{Manuscript.} \\
\text{Person("Anonymous").} \\
\text{reviewed_by("Data on the Web","Anonymous").} \\
\text{Manuscript} \sqcap \exists \text{Reviewed_by.Person} \sqsubseteq \text{Reviewed.}
\end{align*}
\]

by means of the following reasoning:

\[
\begin{align*}
\vdash_{\text{(Type1)}} \text{Manuscript("Data on the Web")} \\
\vdash_{\text{(Type5)}} \text{Reviewed("Data on the Web")}
\end{align*}
\]

Our inference calculus can be used for proving a given Description Logic formula of the selected fragment from an ontology of the same fragment. Our inference calculus can be used for inferring all the entailments from a given ontology. The idea is to apply the rules up to a fix point is reached. In addition, we have designed the inference calculus in order to be implemented in logic programming, in particular, in Prolog. It forces to limit the inference capabilities of our system. The inference calculus only handles the user-defined DL complex formulas (i.e. those included in the TBox). For instance, we cannot infer new relations like $C \sqsubseteq \forall P.D$ because it requires to check all the relations between individuals for $P$ in the ABox. The same can be said for $\exists P.C \sqsubseteq D$, and $P \sqsubseteq Q$.

3 A Query Language based on Description Logic

In this section we will define the query language based on Description Logic. Such query language will introduce variables in DL formulas in order to express
the values to be retrieved in the query result. In addition, our query language can handle conjunctions of DL formulas. We will use variable names starting with lower-case letters to distinguish them from non-variables.

Assuming a set $\mathcal{V}_c$ of variables for concepts $c,d,\ldots$ and a set $\mathcal{V}_p$ of variables for roles $p,q,\ldots$, and a set $\mathcal{V}_i$ of variables for individuals $a,b,\ldots$, a query $Q$ against an ontology $O$ is a conjunction $Q_1,\ldots,Q_n$ where each $Q_i$ has the form:

$Q ::= C \sqsubseteq D \mid E \equiv F \mid R \sqsubseteq T \mid R \equiv T \mid P(A,B) \mid C_0(A)$

where $C \in \mathcal{L}^V$, $D \in \mathcal{R}^V$, $E,F \in \mathcal{E}^V$, $R,T = P,P^-,P^+$ and $P \in \mathcal{P}^V$ and $A,B \in \mathcal{I}^V$, $C_0 \in \mathcal{V}_c$ or is a class name. In addition, $\mathcal{E}^V$ contains the set of formulas of the form:

$\mathcal{L}^V$ contains the set of formulas of the form:

$\mathcal{R}^V$ contains the set of formulas of the form:

and finally, $\mathcal{P}^V$ contains property names and elements of $\mathcal{V}_p$, and $\mathcal{I}^V$ contains individual names and elements of $\mathcal{V}_i$.

As in the case of the data and meta-data definition language, the query language is restricted to a fragment of DL in order to be encoded in logic programming (i.e. in Prolog). Assuming that variable names start with lower case letters, queries are formulas like:
• type(“Growing X Query”) whose meaning is “Find the type of Growing XQuery”.

• Person(p), Reviewed_by (“Growing XQuery”,p), whose meaning is “Retrieve the reviewers of “Growing XQuery”” and

• Manuscript(m), Reviewed_by(m,“Almendros”) whose meaning is “Retrieve the manuscripts in which “Almendros” is a reviewer”

• Author_of (author, “Data on the Web”) whose means is “Retrieve the authors of “Data on the Web””. Let us remark that in this case, subproperties of “Author_of” are taken into account (for instance, “Authored_by” and “Writes”).

Let us remark that the previous queries can be answered from the TBox and the ABox of the ontology. Meta-data can be retrieved by means of our query language, however, queries about meta-data can be only answered from the TBox of the ontology. For instance, using union and intersection operators we can retrieve:

• intersection ≡ Book ⊓ Reviewed whose meaning is “Find the intersections of Book and Reviewed”.

• ⊤ ⊑ ∀ Writes.range whose meaning is “Find the ranges of Writes”.

• ⊤ ⊑ ∀ p.Book whose meaning is “Find the properties whose range is Book”.

• ∃ p.Person ⊑ Book whose meaning is “Find the properties about Person’s in which the range belongs to Book”.

• class ⊑ ∃ p.{”XML”} whose meaning is “Find the classes having a role related to “XML””.

An answer of a query Q w.r.t. an ontology O and vocabulary V is a mapping θ from DVar(Q) into V such that O ⊢ θ(θ(Q)), where DVar(Q) denotes the set of variables of Q.

Finally, we have to remark that some syntactic sugar can be considered in our query language. For instance ∀ P.C(A) represents a(A), a ⊑ ∀ P.C, and ∃ P−. {O}(A) represents a ⊑ ∃ q. {O}, q ⊑ P−, a(A).

4 Encoding into Prolog

Now, we would like to show how to use Prolog in our framework. The role of Prolog is double. Firstly, we can encode any given ontology instance of the considered fragment into Prolog. Secondly, our inference system ⊢ OI can be encoded into Prolog by means of rules, in such a way that a certain class of Prolog goals, which implement the Description Logic based query language can be used as query language.
Table 2
Encoding of the Inference Calculus I

<table>
<thead>
<tr>
<th>Rule Name</th>
<th>Prolog Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Eq1)</td>
<td>triple(E, owl : equivalentClass, E) : ¬class(E).</td>
</tr>
<tr>
<td>(Eq2)</td>
<td>triple(E, owl : equivalentClass, G) : ¬triple(E, owl : equivalentClass, F), triple(F, owl : equivalentClass, G).</td>
</tr>
<tr>
<td>(Eq3)</td>
<td>triple(E, owl : equivalentClass, F) : ¬triple(F, owl : equivalentClass, E).</td>
</tr>
<tr>
<td>(Eq4)</td>
<td>triple(E, owl : equivalentClass, F) : ¬triple(E, rdfs : subclassOf, F), triple(F, rdfs : subclassOf, E).</td>
</tr>
<tr>
<td>(Sub1)</td>
<td>triple(E, rdfs : subclassOf, F) : ¬triple(E, owl : equivalentClass, F).</td>
</tr>
<tr>
<td>(Sub2)</td>
<td>triple(C, rdfs : subclassOf, E) : ¬triple(C, rdfs : subclassOf, D), triple(D, rdfs : subclassOf, E).</td>
</tr>
<tr>
<td>(Sub3)</td>
<td>triple(D, rdfs : subclassOf, E) : ¬triple(union(U), rdfs : subclassOf, E), member(D, U).</td>
</tr>
<tr>
<td>(Sub4)</td>
<td>triple(E, rdfs : subclassOf, C) : ¬triple(E, rdfs : subclassOf, inter(I)), member(C, I).</td>
</tr>
<tr>
<td>(Sub5)</td>
<td>triple(inter(I2), rdfs : subclassOf, D) : ¬triple(inter(I1), rdfs : subclassOf, D), member(C, I1), triple(E, rdfs : subclassOf, C), replace(I1, C, E, I2).</td>
</tr>
<tr>
<td>(Sub6)</td>
<td>triple(union(U2), rdfs : subclassOf, D) : ¬triple(union(U1), rdfs : subclassOf, D), member(C, U1), triple(E, rdfs : subclassOf, C), replace(U1, C, E, U2).</td>
</tr>
<tr>
<td>(Sub7)</td>
<td>triple(C, rdfs : subclassOf, inter(I2)) : ¬triple(C, rdfs : subclassOf, inter(I1)), member(D, I1), triple(D, rdfs : subclassOf, E), replace(I1, D, E, I2).</td>
</tr>
<tr>
<td>(Sub8)</td>
<td>triple(hasvalue(Q, O), rdfs : subclassOf, D) : ¬triple(Q, owl : subclassOf, P), triple(hasvalue(P, O), rdfs : subclassOf, D).</td>
</tr>
<tr>
<td>(Sub9)</td>
<td>triple(exists(Q, C), rdfs : subclassOf, D) : ¬triple(Q, owl : subclassOf, P), triple(exists(P, C), rdfs : subclassOf, D).</td>
</tr>
<tr>
<td>(Sub10)</td>
<td>triple(exists(P, E), rdfs : subclassOf, D) : ¬triple(E, rdfs : subclassOf, C), triple(exists(P, C), rdfs : subclassOf, D).</td>
</tr>
<tr>
<td>(Sub11)</td>
<td>triple(C, rdfs : subclassOf, hasvalue(Q, O)) : ¬triple(P, owl : subclassOf, Q), triple(C, rdfs : subclassOf, hasvalue(P, O)).</td>
</tr>
<tr>
<td>(Sub12)</td>
<td>triple(C, rdfs : subclassOf, forall(Q, D)) : ¬triple(Q, owl : subclassOf, P), triple(C, rdfs : subclassOf, forall(P, D)).</td>
</tr>
<tr>
<td>(Sub13)</td>
<td>triple(C, rdfs : subclassOf, forall(P, E)) : ¬triple(D, rdfs : subclassOf, E), triple(C, rdfs : subclassOf, forall(P, D)).</td>
</tr>
</tbody>
</table>

4.1 Ontology Instance Encoding

The encoding of an ontology instance consists of Prolog facts of a predicate called triple, representing the RDF-triple based representation of OWL. In the case of the TBox: (a) \( en(C \sqsubseteq D) = \text{triple}(en(C), rdfs : \text{subclassOf}, -en(D)) \); (b) \( en(E \equiv F) = \text{triple}(en(E), owl : \text{equivalentClass}, en(F)) \); (c)
en(P ⊆ Q) = triple(en(P), rdfs : subPropertyOf, en(D)); and (d) en(P ⊑ Q) = triple(en(P), owl : equivalentProperty, en(Q)). In addition, en(C), en(D), en(E), en(F), en(P) and en(Q) represents the encoding of classes and properties in which class and property names C, P, . . . are translated as Prolog atoms c, d, . . . The special case of ⊤ is encoded as en(⊤) = owl : thing. In addition, Prolog terms are used for representing complex DL formulas as follows: (a) en(P+) = trans(en(P)); (b) en(P−) = inv(en(P)); (c)
\( \forall P, C \) = \text{forall}\((\text{en}(P), \text{en}(C))\); (d) \( \text{en}(\exists P, C) = \text{exists}\((\text{en}(P), \text{en}(C))\); (e) \( \text{en}(\exists P, \{O\}) = \text{hasvalue}\((\text{en}(P), \text{en}(O))\); (f) \( \text{en}(\exists P, C) = \text{hasvalue}\((\text{en}(P), \text{en}(C))\); (g) \( \text{en}(\exists P, C) = \text{hasvalue}\((\text{en}(P), \text{en}(C))\); (h) \( \text{en}(\exists P, C) = \text{hasvalue}\((\text{en}(P), \text{en}(C))\); (i) \( \text{en}(\exists P, C) = \text{hasvalue}\((\text{en}(P), \text{en}(C))\); (j) \( \text{en}(\exists P, C) = \text{hasvalue}\((\text{en}(P), \text{en}(C))\).

4.2 Encoding of the Inference Calculus \( \vdash_{O1} \)

Now, the second element of the encoding consists of Prolog rules for encoding the \( \vdash_{O1} \) inference calculus. The set of rules can be found in Tables 2, 3 and 4 where facts for predicates class, property and individual are defined for each atomic classes, (inverse and transitive) properties and individuals.

4.3 Using Prolog as Query Language

In this section, we will show how to use Prolog as query language for OWL. Basically, each query \( \phi = \varphi_1, \ldots, \varphi_n \) in our query language can be encoded as a Prolog goal \( ? - \text{triple}(\text{en}(\varphi_1)), \ldots, \text{triple}(\text{en}(\varphi_n)) \) in which each element of \( V_c, V_p \) and \( V_i \) is encoded as a Prolog Variable. Now, we will show examples of queries against the ontology \( O_0 \) and the corresponding answers. Let us remark that in Prolog variables start with upper case letters.

**Query 1:** The first query we like to show is “Retrieve the authors of manuscripts”, which can be expressed in our query language as \( \text{Author _of}(a, b), \text{Manuscript}(b) \). It can be encoded as:

\[
? - \text{triple}(A, \text{author _of}, B), \\
\text{triple}(B, \text{rdf : type}, \text{manuscript}).
\]

Let us remark that our inference system is able to infer that a Paper and a Book is a Manuscript and therefore the above query retrieves all the manuscripts of the ontology \( O_0 \). In addition, our inference system is able to infer that author _of is a equivalent property to writes, and the inverse of authored_by, and therefore all the cases are retrieved by the query language. In this case, the answers will be "Abiteboul", "Buneman", "Suciu", "Simeon", "Buneman", together with names of manuscripts.

**Query 2:** The second query we would like to show is “Retrieve the books of topic XML” which can be expressed as \( \text{Book}(book), \text{topic}(book," \text{XML}") \). Now, it can be expressed as:

\[
? - \text{triple}(Book, \text{rdf : type}, book), \\
\text{triple}(Book, \text{topic},” \text{XML}”).
\]
However, given that the ontology already includes the class “XMLBook” we can express the same query in a more concise way as $XMLBook(book)$ and therefore as:

$$ ? \triangleright triple(Book, rdf : type, xmlbook) $$

**Query 3:** The third query we would like to show is “Retrieves the writers of reviewed manuscripts”. It can be expressed as $Reviewed(manuscript), Writes(author, manuscript)$. In this case, the query can be expressed as:

$$ ? \triangleright triple(Manuscript, rdf : type, reviewed),
\quad triple(Author, writes, Manuscript). $$

**Query 5:** Let us see an example of query for retrieving meta-data from the ontology. For instance $range \sqsubseteq \forall Reviewed\_by.domain$ whose meaning is “Find the domain and range of Reviewed\_by” is encoded as:

$$ ? \triangleright triple(Range, rdfs : subClassOf).
\quad \forallall(reviewed\_by, Domain)). $$

In this case the answers will be Manuscript, Book, Paper for Domain and Man, Woman, Person for Range.

## 5 Conclusions and Future Work

In this paper we have proposed a query language for OWL based on Prolog. The query language is able to query about data and meta-data of a given ontology. As future work, we would like to extend our approach to richer fragments of DL which has been studied in [Vol04]. We believe that some extensions could be possible following the same technique here presented. In addition, we would like to extend our work in the line of SWRL by incorporating Prolog rules to the reasoning with OWL, and therefore for enriching the defined query language.

## References


