A Prolog library for OWL RL

Jesús M. Almendros-Jiménez
Dpto. de Lenguajes y Computación
Universidad de Almería
04120-Spain
jalmen@ual.es

ABSTRACT
In this paper we describe the development of a Prolog library for OWL RL. OWL RL has been recently proposed by the W3C consortium as a fragment of OWL 2 for which reasoning can be done efficiently. In this context, we have studied how to implement a Prolog library based on OWL RL. By means of Prolog rules we are able to infer new knowledge from a given ontology. The OWL RL library has been implemented under the SWI-Prolog interpreter and is based on the RDF library provided by the SWI-Prolog environment, in such a way that OWL triples are computed and stored in secondary memory.

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) [KC04] and the Web Ontology Language (OWL) [Gro09]. 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) [BG04] 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) 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 fragments of Description Logic are subsets or variants of C2, 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 main 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 [MPSG09], in which a direct semantics is defined based on Description Logic, and in [Sch09] a RDF-based semantics is provided.

In this paper we describe the development of a Prolog library for OWL RL. OWL RL has been recently proposed by the W3C consortium as a fragment of OWL 2 for which reasoning can be done efficiently. In this context, we have studied how to implement a Prolog library based on OWL RL. By means of Prolog rules we are able to infer new knowledge from a given ontology. The OWL RL library has been implemented under the SWI-Prolog interpreter and is based on the RDF library provided by the SWI-Prolog environment, in such a way that OWL triples are computed and stored in secondary memory.

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 XQuery 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].

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) hierarchies (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.

LID 2011 March 25, 2011, Uppsala, Sweden
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 [SPG+07]) are based on tableaux based decision procedures. Based on logic programming, Description Logic Programming [GHVD03] has been presented as the intersection of Description Logic and Logic Programming. Basically, the framework identifies a subset of DL which can be mapped into Logic Programming. It is the basis of some tools like KAON2 tool [BEH+02, HMS08] and Sweet Prolog [LTD04]. The DLog system [LSK08] reasons with an ontology by means of the encoding into Prolog and the use of the PTTP theorem prover.

On the other hand, 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 rdfs:type, rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain and rdfs:range.

In the case of OWL, the so-called $pD^*$ approach [H04] 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.

Recently, the W3C has proposed the new OWL 2. The OWL 2 specification provides OWL profiles that correspond to syntactic subsets of the OWL 2 language, namely OWL EL, OWL QL and OWL RL. Each profile is designed to trade some expressive power for efficiency of reasoning. For example, the OWL EL profile trades expressivity for the benefit of polynomial time subsumption testing. Similarly, reasoning for the OWL RL profile can be implemented using a rule engine. Finally, OWL 2 QL has been defined as basis for an efficient query language.

OWL 2 RL is aimed at applications that require scalable reasoning without sacrificing too much expressive power. It is designed to accommodate OWL 2 applications that can trade the full expressivity of the language for efficiency, as well as RDF(S) applications that need some added expressivity. OWL 2 RL reasoning systems can be implemented using rule-based reasoning engines. The ontology consistency, class expression satisfiability, class expression subsumption, instance checking, and conjunctive query answering problems can be solved in time that is polynomial with respect to the size of the ontology. The design of OWL 2 RL was inspired by Description Logic Programs [GHVD03] and $pD^*$ [H04].

The aim of our work is the use of Prolog for the implementation of the OWL RL profile. Such implementation provides a Prolog library to be used for OWL 2 inference, and therefore reasoning and querying. In particular, the OWL 2 profile provides certain rules which can be used for ontology consistency checking. In order to implement the Prolog library we have adopted the following decisions:

- Firstly, the OWL RL rules are recursive in most of cases, and moreover, some of them loops in a Prolog interpreter. Therefore, we have defined a bottom-up interpreter in Prolog in order to avoid infinite computations. The bottom-up interpreter evaluates OWL RL rules from right to left and memorizes computations in such a way that it avoids the call to elements previously computed.

- Secondly, the application of the OWL RL Prolog rules to a given ontology obtains a new ontology containing the inferred elements which are stored in secondary memory. Secondary memory is also used for storing the initial ontology and intermediate computations when applying OWL RL Prolog rules. In particular, secondary memory is used for memorization of computations.

- Thirdly, since bottom-up computation is required, Prolog can be used for querying the new ontology after the new ontology is computed. In other words, the Prolog library works as follows: given a certain ontology, the user calls the Prolog library to compute the new triples of the ontology. Once they have been computed and stored in secondary memory, the user might call the store for querying the ontology.
We have developed a prototype of our approach which can be downloaded from http://indalog.ual.es/OWL-RL-Prolog.

Figure 1: OWL RL [MGH+09]

<table>
<thead>
<tr>
<th>TBox</th>
<th>Allo</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Man(“Abiteboul”)</td>
<td>(1) Top(average, rating)</td>
</tr>
<tr>
<td>(2) Man(“Buneman”)</td>
<td>(2) Paper(“Growing XQuery”)</td>
</tr>
<tr>
<td>(3) Man(“Suciu”)</td>
<td>(3) Book(“XML in Scotland”)</td>
</tr>
<tr>
<td>(4) Book(“Data on the Web”)</td>
<td>(4) Person(“Anonymous”)</td>
</tr>
<tr>
<td>(6) Paper ⊑ ∪ Book ⊑ Manuscript</td>
<td>(6) Manuscript ∩ ∃reviewed_by,Person ⊑ Reviewed</td>
</tr>
<tr>
<td>(8) Manuscript ∩ ∃reviewed_by,Person ⊑ Reviewed</td>
<td>(8) Person ⊑ ∃average_rating, ⊑ rating</td>
</tr>
<tr>
<td>(9) OneAuthor ⊑ ∃author_of, Person</td>
<td>(9) Manuscript ⊑ ∃rating.Score</td>
</tr>
<tr>
<td>(10) Manuscript ⊑ ∃rating.Score</td>
<td>(10) Person ⊑ ∃average_rating, ⊑ rating</td>
</tr>
<tr>
<td>(11) Manuscript ⊑ ∀topic.Topic</td>
<td>(11) Person ⊑ ∃average_rating, ⊑ rating</td>
</tr>
<tr>
<td>(12) average_rating ⊑ rating</td>
<td>(12) Manuscript ⊑ ∃rating.Score</td>
</tr>
<tr>
<td>(13) author_of ≡ writes</td>
<td>(13) average_rating ⊑ rating</td>
</tr>
<tr>
<td>(14) authored_by ≡ author_of</td>
<td>(14) Man(“Suciu”)</td>
</tr>
<tr>
<td>(15) T ⊑ ∀ author_of, Manuscript</td>
<td>(15) Woman ⊑ Person</td>
</tr>
<tr>
<td>(16) T ⊑ ∀ author_of, Person</td>
<td>(16) Paper ⊑ ∪ Book ⊑ Manuscript</td>
</tr>
<tr>
<td>(17) T ⊑ ∃ reviewed_by, Person</td>
<td>(17) Paper ⊑ ∪ Book ⊑ Manuscript</td>
</tr>
<tr>
<td>(18) T ⊑ ∃ reviewed_by, Manuscript</td>
<td>(18) Manuscript ⊑ ∃rating.Score</td>
</tr>
<tr>
<td>(19) T ⊑ ∀ friendOf, Person</td>
<td>(19) Manuscript ⊑ ∃rating.Score</td>
</tr>
<tr>
<td>(20) T ⊑ ∀ friendOf, Person</td>
<td>(20) Manuscript ⊑ ∃rating.Score</td>
</tr>
<tr>
<td>(21) first_reader ≡ friendOf, o author_of</td>
<td>(21) Manuscript ⊑ ∃rating.Score</td>
</tr>
<tr>
<td>(22) friendOf ≡ friendOf</td>
<td>(22) Manuscript ⊑ ∃rating.Score</td>
</tr>
<tr>
<td>(23) Punct(average, rating)</td>
<td>(23) Manuscript ⊑ ∃rating.Score</td>
</tr>
</tbody>
</table>

Figure 2: An Example of Ontology

<table>
<thead>
<tr>
<th>Subclass Expressions</th>
<th>Superclass Expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>a class other than owl:Thing</td>
<td>a class other than owl:Thing</td>
</tr>
<tr>
<td>an enumeration of individuals (owl:oneOf)</td>
<td>intersection of classes (owl:intersectionOf)</td>
</tr>
<tr>
<td>intersection of class expressions (owl:intersectionOf)</td>
<td>negation (owl:complementOf)</td>
</tr>
<tr>
<td>union of class expressions (owl:unionOf)</td>
<td>universal quantification to a class and range (owl:allValuesFrom)</td>
</tr>
<tr>
<td>existential quantification to a class and range (owl:someValuesFrom)</td>
<td>existential quantification to an individual and literal (owl:hasValue)</td>
</tr>
<tr>
<td>to an individual and literal (owl:hasValue)</td>
<td>at-most 0/1 cardinality restriction</td>
</tr>
<tr>
<td>to a class and range (owl:maxCardinality)</td>
<td></td>
</tr>
</tbody>
</table>

• Fourthly, we have used the SWI-Prolog interpreter for the implementation of the library. Fortunately, SWI-Prolog provides a RDF library [WHV08] which allows to store the RDF graph by means of secondary memory.

• Finally, OWL RL Prolog rules for consistency checking have been integrated and they provide warning messages when a certain inconsistency has been detected. Such warning messages are obtained during the inference process. In the case of inconsistency, the user might modify the current ontology and after (s)he might call the Prolog library for re-computing the elements of the new ontology.

Let us remark that in SWI-Prolog there exists a Prolog library called Thela2 [VWM09] to support OWL 2 following precisely its structural syntax specification: every axiom in the ontology would correspond on a one-to-one basis with facts in the Prolog database. This allows querying the Ontology by simply query the Prolog database using goals with variables as arguments. However, as far as we know, OWL RL reasoning is not part of the library.

We have tested our prototype with several examples of ontologies including the running example presented below.

The structure of the paper is as follows. Section 2 will present the elements of OWL RL. Section 3 will describe the Prolog library of OWL RL. Section 4 will show some examples of use of the Prolog library. Finally, Section 5 will conclude and present future work.

2. OWL RL

In this section, we will present the OWL 2 profile called OWL RL. The reader can check the official specification in [MGH+09]. Basically, OWL RL is a fragment of OWL 2 with several restrictions in the use of the vocabulary. The table 1 summarizes OWL RL. Thus, OWL 2 RL supports all axioms of OWL 2 apart from disjoint unions of classes (owl:DisjointUnion) and reflexive object property axioms (owl:ReflexiveProperty).

Let us see an example of an ontology (see Figure 2). The ontology describes meta-data in the TBox defining that Person’s are Animal’s and equivalent to Human’s (axioms (1) and (2)); the elements of Man and the elements of Woman are elements of Person (axioms (3) and (4)); and the
ments of Paper and elements of Book are elements of Manuscript (axiom (6)). In addition, a Writer is a Person who is the author_of a Manuscript (axiom (5)), and the class Reviewed contains the elements of Manuscript reviewed_by a Person (axiom (8)). Moreover, the XMLBook class contains the elements of Manuscript which have as topic the value “XML” (axiom (7)) and the class OneAuthor is a subclass of elements authored_by at most one Person (axiom (9)). The classes Score and Topic contain, respectively, the values of the properties rating and topic associated to Manuscript (axioms (10) and (11)). The property average_rating is a suproperty of rating (axiom (12)). The property writes is equivalent to author_of (axiom (13)), and authored_by is the inverse property of author_of (axiom (14)). Moreover, the property author_of, and conversely, reviewed_by, has as domain a Person and as range a Manuscript (axioms (15)-(18)). The relation friendOf is defined between Person’s and is symmetric (axioms (19),(20) and (22)), and average_rating is a functional property (axiom (24)). Finally, (axiom (21)) the property first_reader is the property composition of friendOf and author_of.

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. Finally, Buneman and Abiteboul are friends, and Almendros is the same as Jesus.

OWL RL is intended to be used for inferring information from a given ontology. As an example of inference, we can consider the computation of Reviewed and Writer classes which, in the previous ontology, would include Data on the Web, and Growing XQuery for the case of class Reviewed, and Abiteboul, Buneman, Suciu and Simeon for the case of class Writer. Let us remark that the inference has to use that Paper’s and Book’s are Manuscript’s, and the relationships between author_of, authored_by and writes given that they are in some sense equivalent. Finally, Abiteboul is a first_reader of XML in Scotland, since he is a friend of the author of such book (Buneman is also first_reader of Data on the Web). As an example of consistency checking, the average_rating property has been defined as functional, and therefore no more than one different value has to be assigned (let us remark that it does not happen in the example shown).

### 3. PROLOG LIBRARY FOR OWL RL

Now, we would like to show the elements of the Prolog library. Firstly, we summarize the main predicates of the library.

- The main predicate is load(+FileName) which carries out the inference and is defined as follows:
  ```prolog
  load(File):-rdf_reset_db,  
  rdf_load(File),  
  bottom_up_main.
  ```

  Basically, the predicate cleans the RDF database by means of rdf_reset_db, loads the file into the database by means of rdf_load, and calls the bottom-up interpreter.

- The main predicate of the bottom-up interpreter is defined as follows:
  ```prolog
  bottom_up_main:-bottom_up_procedure,fail.  
  bottom_up_main.
  ```

  which always succeeds. It calls the predicate bottom_up_procedure which is defined as follows:

  ```prolog
  bottom_up_procedure:-bottom_up_init,fail.  
  bottom_up_procedure:-assert(end),bottom_up_step-bottom_up_procedure.  
  ```

  The predicate bottom_up_procedure is executed in two stages. The first stage initiates the bottom-up interpreter by means of bottom_up_init. The second stage asserts a flag (end) which serves as bottom-up control, and calls the bottom-up loop (bottom_up_step). The predicate bottom_up_init is defined as follows:

  ```prolog
  bottom_up_init:-  
  retractall(trp_store(_,_,_)),fail.  
  bottom_up_init:-rdf(X,GT,Y),  
  rdf_global_term(Y,GY),  
  assert(trp_store(X,Y,Z)),fail.  
  bottom_up_init:-rdf_reset_db,fail.  
  bottom_up_init.
  ```

  In each step of the bottom-up mechanism the new triples obtained from the ontology and the OWL RL rules are temporarily stored in main memory by means of a dynamic Prolog predicate called trp_store. The bottom-up mechanism starts adding to main memory the triples of the source file \(^2\). The triples of the source file have been stored in secondary memory by means of the rdf_load predicate, and they can be retrieved by means of calls to the RDF SWI-Prolog predicate rdf. The predicate bottom_up_step is defined as follows:

  ```prolog
  bottom_up_step:-clean,update,bottom_up_rule,  
  (end->bottom_up_step;fail).  
  bottom_up_step.
  ```

  The predicate clean removes the flag end, and update updates the RDF database with the new elements obtained from the ontology. In the first step, they correspond with the triples of the source file. In each step, all the OWL RL are applied, and whenever the flag end has been added, the bottom-up loop continues. The flag end is added when at least a new triple has been added by means of the OWL RL rules. Finally, each rule is applied from right to left by means of the predicate bottom_up_rule, defined as follows:

  ```prolog
  bottom_up_rule:-clause(triple(X,Y,Z,C),call_condition(C),  
  (end->bottom_up_step;fail).  
  bottom_up_step.
  ```

  \(^2\)Let us remark that rdf_global_term is a RDF Prolog predicate for translating an alias and local name into a global name.
 basically, the application of a OWL RL rule is as follows. The Prolog rules of OWL RL are rules of the form triple():-triple(),...triple(). the condition of the rule is evaluated and whenever the rule generates a new triple, the flag end is added, and the new triple is asserted into the main memory. the call_condition predicate is defined as follows:

```
call_condition(true):-!.
call_condition(C):-C=[',|Conditions],!,
call_list(Conditions).
call_condition(C):-C=triple(X,Y,Z),!,
    rdf_global_term(Y,GY),
    rdf(X,GY,Z).
call_condition(C):-call(C).
call_list([]).
call_list([C|RC]):=call_condition(C),
call_list(RC).
```

When a rule condition calls to the triple predicate, the bottom-up interpreter calls the rdf predicate in order to retrieve the element from secondary memory. When rdf fails, it means that the rule cannot be applied.

3.1 Prolog Rules for OWL RL

Now, we would like to show how the OWL RL rules are represented in Prolog. Figure 3 shows some of the rules of the Prolog library. The complete set of rules can be downloaded from http://indalog.ual.es/OWL-RL-Prolog.

As was commented before, the Prolog rules for OWL RL have the form triple():-triple(),...triple(). For instance, rule (1) defines the transitive closure of the subclass relationship. Such a rule loops in a Prolog interpreter, however, with our bottom-up implementation, when a call to the predicate triple is achieved in the rule condition, the interpreter retrieves the instances from the RDF database, and it does not call to the rules of triple. Therefore, when the instances of both atoms of the rule condition are already in the RDF database, a new triple is generated and stored in the RDF database. For instance, triple(#Man, rdfs:subClassOf, #Animal) is obtained whenever triple(#Man, rdfs:subClassOf, #Person) and triple(#Person, rdfs:subClassOf, #Animal) are already in the database. Rules (2)-(4) combine typing with the subclass relationship and the domain/range definitions. Rules (5)-(7) add new triples from property relationships. Rules (8) and (9) add a new type triple from a quantified DL formula. Finally, Rule (10) handles a set operator in DL, in this case, intersection. Fortunately, the SWI-Prolog library for RDF provides a predicate rdfs_member for the handling of RDF lists.

3.2 Consistency Checking

Now, we would like to show how the Prolog library handles consistency checking of the OWL RL ontologies. The OWL RL specification [MGH+09] gives us a set of rules for consistency checking. In our implementation, we have defined Prolog rules for consistency checking which are similar to the previously presented. Basically, they are rules with head triple(warning,warning,warning). The bottom-up interpreter generates such a triple whenever an inconsistency is found. In addition, the same rule generates a message as output giving details about the found error. For instance, let us suppose we say that a certain book has as average rating, which is a functional property, two values: Good and Excellent but they are defined as different. Let us remark that they have to be declared to be different (i.e. by means of owl:differentFrom), because otherwise there is a rule that makes them equal by means of owl:sameAs. In such a case we would obtain the following message:
4. ONTOLOGY QUERYING

Now, we would like to show how to query the ontology using Prolog. The advantages of using Prolog is that we are equipped with a programming language with many primitives and libraries for programming queries against the OWL database. Let us see some examples.

Firstly, we have to give some hints about how to use the Prolog library. In order to handle name-spaces by means of the RDF library of SWI-Prolog we have to register the name space by means of the predicate rdf_db:ns. For instance, our example has been created by means of the Protege tool [GMF+03] which has assigned a name and we can assign ex to it in SWI-Prolog by means of the following command line call: rdf_db:ns(ex, 'http://www.semanticweb.org/ontologies/2010/6/Ontology1278090045298.owl#'). Now, we can use from SWI-Prolog the name ex.

After, we can load the OWL file by means of load('example_owl_rl.owl'), assuming that our example is stored in such a file. Once the file is loaded we can now make queries against the RDF database which has stored the complete set of triples obtained from the source file.

Query 1: The first query we like to show is “Retrieve the authors of manuscripts”, which can be expressed in Prolog as:

?- rdf(X,ex:author_of,Y),
   rdf(Y,rdf:type,ex:'Manuscript').

Let us remark that the OWL RL rules are able to infer that a Paper and a Book is a Manuscript and therefore the above query retrieves all the manuscripts of the ontology. In addition, they are 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 means of Prolog. In this case, the answer is:

X = 'http://www.semanticweb.org...#Abiteboul',
Y = 'http://www.semanticweb.org...#DataontheWeb' ;
X = 'http://www.semanticweb.org...#Abiteboul',
Y = 'http://www.semanticweb.org...#XMLinScotland' ;
X = 'http://www.semanticweb.org...#Suciu',
Y = 'http://www.semanticweb.org...#DataontheWeb' ;
X = 'http://www.semanticweb.org...#Simeon',
Y = 'http://www.semanticweb.org...#GrowingXQuery' ;
X = 'http://www.semanticweb.org...#Buneman',
Y = 'http://www.semanticweb.org...#DataontheWeb' ;

Query 2: The second query we would like to show is “Retrieve the books of topic XML” which can be expressed as:

?- rdf(X,rdf:type,ex:'Book'),
   rdf(X,ex:topic_of,ex:'XML').

However, given that the ontology already includes the class “XMLBook” we can express the same query in a more concise way as:

?- rdf(X,rdf:type,ex:'XMLBook')

Query 3: The third query we would like to show is “Retrieve the writers of reviewed manuscripts”. It can be expressed as:

?- rdf(X,rdf:type,ex:'Writer'),
   rdf(X,ex:author_of,Y),
   rdf(Y,rdf:type,ex:'Reviewed').

obtaining as answer:

X = 'http://www.semanticweb.org...#Abiteboul',
Y = 'http://www.semanticweb.org...#DataontheWeb' ;
X = 'http://www.semanticweb.org...#Suciu',
Y = 'http://www.semanticweb.org...#DataontheWeb' ;
X = 'http://www.semanticweb.org...#Simeon',
Y = 'http://www.semanticweb.org...#GrowingXQuery' ;

Query 4: Let us now suppose that we would like to retrieve the scores of the ontology in a list. It can be expressed as follows and we obtain the answer below:

?- findall(X,rdf(X,rdf:type,ex:'Score'),L).
L = ['http://www.semanticweb.org...#Excellent',
     'http://www.semanticweb.org...#Good'].

Query 5: The next query is about meta-data. For instance, we can ask about the sub-properties of a given property, obtaining:

?- rdf(X,rdfs:subPropertyOf,ex:author_of).
X = 'http://www.semantic...#author_of' ;
X = 'http://www.semantic...#writes'.

Query 6: Finally, we can collect the set of (distinct) equivalent properties as follows:

?- findall(X,Y),(rdf(X,owl:equivalentProperty,Y), X\=Y),L.
L = ['http://www.semantic...#author_of',
     'http://www.semantic...#writes'],
     ('http://www.semantic...#writes',
     'http://www.semantic...#author_of')].

Finally, we can save all the inferred triples by means of the RDF SWI-Prolog primitive rdf_save(+FileName).
5. CONCLUSIONS AND FUTURE WORK
In this paper we have described the development of a Prolog library for OWL RL. By means of Prolog rules we are able to infer new knowledge from a given ontology and detect ontology consistency. The OWL RL library has been implemented under the SWI-Prolog interpreter and is based on the RDF library provided by the SWI-Prolog environment, in such a way that OWL triples are computed and stored in secondary memory. As future work, we would like to study how to implement other relevant fragments of OWL 2 in Prolog. In particular, we are interested in implementing OWL QL which has been defined as suitable fragment for conjunctive queries against OWL resources. The differences w.r.t. our approach is that OWL QL requires query rewriting in order to implement it.

6. REFERENCES


