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 *HMTL* 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* [1] 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 $\mathcal{C}^2$, 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 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 $\mathcal{SROIQ}$ which is the basis of the new standarized 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 *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].

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.

*The author work has been partially supported by the Spanish MICINN under grant TIN2008-06622-C03-03.
[1] http://www.w3.org.

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* [LTBS04]. 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 `rdf:type`, `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:domain` and `rdfs:range`.

In the case of 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.

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 a 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^* [tH04].

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.

Figure 1: OWL RL [MGH+09]

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

Figure 2: An Example of Ontology

TBox	
(1) $Person \sqsubseteq Animal$	*(2)* $Person \equiv Human$
(3) $Man \sqsubseteq Person$	*(4)* $Woman \sqsubseteq Person$
(5) $Person \sqcap \exists author\_of.Manuscript \sqsubseteq Writer$	*(6)* $Paper \sqcup Book \sqsubseteq Manuscript$
(7) $Book \sqcap \exists topic.\{\text{“}XML\text{”}\} \sqsubseteq XMLbook$	*(8)* $Manuscript \sqcap \exists reviewed\_by.Person \sqsubseteq Reviewed$
(9) $OneAuthor \sqsubseteq \exists authored\_by_{\leq 1}.Person$	*(10)* $Manuscript \sqsubseteq \forall rating.Score$
(11) $Manuscript \sqsubseteq \forall topic.Topic$	*(12)* $average\_rating \sqsubseteq rating$
(13) $author\_of \equiv writes$	*(14)* $authored\_by \equiv author\_of^{-}$
(15) $\top \sqsubseteq \forall\ author\_of.Manuscript$	*(16)* $\top \sqsubseteq \forall\ author\_of^{-}.Person$
(17) $\top \sqsubseteq \forall\ reviewed\_by.Person$	*(18)* $\top \sqsubseteq \forall\ reviewed\_by^{-}.Manuscript$
(19) $\top \sqsubseteq \forall\ friendOf.Person$	*(20)* $\top \sqsubseteq \forall\ friendOf^{-}.Person$
(21) $first\_reader \equiv friendOf\ o\ author\_of$	*(22)* $friendOf \equiv friendOf^{-}$
(23) Funct(average_rating)	
ABox	
(1) Man(“Abiteboul”)	
(2) Man(“Buneman”)	*(3) Man(“Suciu”)*
(4) Book(“Data on the Web”)	*(5) Book(“XML in Scottland”)*
(6) Paper(“Growing XQuery”)	*(7) Person(“Anonymous”)*
(8) author_of(“Abiteboul”,“Data on the Web”)	*(9) authored_by(“Data on the Web”,“Buneman”)*
(10) author_of(“Suciu”,“Data on the Web”)	*(11) author_of(“Buneman”,“XML in Scottland”)*
(12) writes(“Simeon”,“Growing XQuery”)	*(13) reviewed_by(“Data on the Web”,“Anonymous”)*
(14) reviewed_by(“Growing XQuery”,“Almendros”)	*(15) average_rating(“Data on the Web”,“Good”)*
(16) rating(“XML in Scottland”,“Excellent”)	*(17) average_rating(“Growing XQuery” ,“Good”)*
(18) topic(“Data on the Web”,“XML”)	*(19) topic(“Data on the Web”,“Web”)*
(20) topic(“XML in Scottland”,“XML”)	*(21) friendOf(Abiteboul,Buneman)*
(22) Almendros ≡ Jesus	

- 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 *Thea2* [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 developed a prototype of our approach which can be downloaded from `http://indalog.ual.es/OWL-RL-Prolog`. We have tested our prototype with several examples of ontologies including the running example presented bellow.

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 ele-

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 subproperty 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 conversively, *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 Scottland" 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 Scottland*, 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:

```
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:

```
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:

```
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:

```
bottom_up_init:-
           retractall(trp_store(_,_,_)),fail.
bottom_up_init:-rdf(X,GY,Z),
             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:

```
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:

```
bottom_up_rule:-clause(triple(X,Y,Z),C),
                call_condition(C),
```

[2]Let us remark that `rdf_global_term` is a RDF Prolog predicate for translating an alias and local name into a global name.

Figure 3: OWL RL in Prolog

```
(1) triple(C1, rdfs:subClassOf, C3):-triple(C1, rdfs:subClassOf, C2),triple(C2, rdfs:subClassOf, C3).
(2) triple(X, rdf:type, C2):-triple(C1, rdfs:subClassOf, C2),triple(X, rdf:type, C1).
(3) triple(X, rdf:type, C):-triple(P, rdfs:domain, C),triple(X, P, _).
(4) triple(Y, rdf:type, C):-triple(P, rdfs:range, C),triple(_, P, Y).
(5) triple(Y, P, X):-triple(P, rdf:type,  'http://www.w3.org/2002/07/owl#SymmetricProperty'),triple(X, P, Y).
(6) triple(X,P2,Y):-triple(P1, rdfs:subPropertyOf, P2),triple(X,P1,Y).
(7) triple(X,P2,Y):-triple(P1, owl:equivalentProperty, P2), triple(X,P1,Y).
(8) triple(U, rdf:type, X):-triple(X, owl:someValuesFrom, Y),
                      triple(X, owl:onProperty, P),triple(U,P,V),triple(V, rdf:type, Y).
(9) triple(U,P,Y):-triple(X, owl:hasValue,Y),triple(X, owl:onProperty,P),triple(U, rdf:type, X).
(10) triple(C, rdfs:subClassOf, D):-triple(C, owl:intersectionOf, X),rdfs_member(D,X).
```

```
                    rdf_global_term(Y,GY),
                    (rdf(X,GY,Z)->fail;
                    (\+end,X\=literal(_)->
                    assert(end);
                    true),
                   (X\=literal(_)->
                   assert(trp_store(X,Y,Z));
                   true),
                   fail).
bottom_up_rule.
```

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 handle 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 *Excelent* 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:

```
?- load('example_owl_rl.owl').
% Parsed "example_owl_rl.owl" in 0.01 sec; 154 triples
Warning: Same and Different Individuals
#Excellent
#Good
```

Let us now assume that `Data on the Web` has been declared of type `OneAuthor`, and `Abiteboul`, `Buneman` and `Suciu` are different individuals, then we obtain the following message:

```
Warning: Same and Different Individuals
#Suciu
#Buneman
```

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 *Protegé* 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...#XMLinScottland' ;
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...#Buneman',
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

[ABE06] J. M. Almendros-Jiménez, A. Becerra-Terón, and Francisco J. Enciso-Baños. Magic sets for the XPath language. *Journal of Universal Computer Science*, 12(11):1651–1678, 2006.

[ABE08] J. M. Almendros-Jiménez, A. Becerra-Terón, and Francisco J. Enciso-Baños. Querying XML documents in logic programming. *Theory and Practice of Logic Programming*, 8(3):323–361, 2008.

[ABE09] J. M. Almendros-Jiménez, A. Becerra-Terón, and F. J. Enciso-Baños. Integrating XQuery and Logic Programming. In *Proceedings of the 17th International Conference on Aplications of Declarative Programming and Knowledge Management, INAP'07 and 21th Workshop on (Constraint) Logic Programming, WLP'07*, pages 117–135, Heidelberg, Germany, 2009. Springer LNAI, 5437.

[Alm08] J. M. Almendros-Jiménez. An RDF Query Language based on Logic Programming. In *Proceedings of the 3rd Int'l Workshop on Automated Specification and Verification of Web Systems*. Electronic Notes on Theoretical Computer Science (200), 67–85, 2008.

[Alm09a] J. M. Almendros-Jiménez. An Encoding of XQuery in Prolog. In *Procs of the Sixth International XML Database Symposium XSym'09, at VLDB'09*, pages 145–155, Heidelberg, Germany, 2009. Springer, LNCS 5679.

[Alm09b] J. M. Almendros-Jiménez. Extending XQuery for Semantic Web Reasoning. In *Procs of the International Conference on Applications of Declarative Programming and Knowledge Management, INAP'09*, pages 109–124, 2009.

[Alm09c] J. M. Almendros-Jiménez. Ontology Querying and Reasoning with XQuery. In *Proceedings of the PLAN-X 2009: Programming Language Techniques for XML, An ACM SIGPLAN Workshop co-located with POPL 2009*. `http://db.ucsd.edu/planx2009/papers.html`, 2009.

[BCM+03] F. Baader, D. Calvanese, D.L. McGuinness, P. Patel-Schneider, and D. Nardi. *The description logic handbook: theory, implementation, and applications*. Cambridge Univ Press, 2003.

[BEH+02] Erol Bozsak, Marc Ehrig, Siegfried Handschuh, Andreas Hotho, Alexander Maedche, Boris Motik, et al. KAON - Towards a Large Scale Semantic Web. In *E-commerce and Web technologies: Third International Conference, EC-Web 2002*, pages 304–313. Springer, 2002.

[BG04] Dan Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. Technical report, `http://www.w3.org/TR/rdf-schema/`, 2004.

[BLHL+01] T. Berners-Lee, J. Hendler, O. Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[EIKP08] T. Eiter, G. Ianni, T. Krennwallner, and A. Polleres. Rules and ontologies for the semantic web. In *Reasoning Web*, pages 1–53. Springer, LNCS 5224, 2008.

[GHVD03] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proc. of the International Conference on World Wide Web*, pages 48–57, NY, USA, 2003. ACM Press.

[GKV97] E. Grädel, P.G. Kolaitis, and M.Y. Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997.

[GMF+03] J.H. Gennari, M.A. Musen, R.W. Fergerson, W.E. Grosso, M. Crubézy, H. Eriksson, N.F. Noy, and S.W. Tu. The evolution of Protégé: an environment for knowledge-based systems development. *International Journal of Human-Computer Studies*, 58(1):89–123, 2003.

[Gro09] W3C Working Group. OWL 2 Ontology Web Language. Technical report, `http://www.w3.org/TR/owl2-overview/`, 2009.

[HHP08] A. Hogan, A. Harth, and A. Polleres. SAOR: Authoritative Reasoning for the Web. In *Proceedings of the 3rd Asian Semantic Web Conference on The Semantic Web*, pages 76–90. Springer-Verlag, 2008.

[HMS08] U. Hustadt, B. Motik, and U. Sattler. Deciding expressive description logics in the framework of resolution. *Information and Computation*, 206(5):579–601, 2008.

[HMW08] V. Haarslev, R. Möller, and S. Wandelt. The revival of structural subsumption in tableau-based description logic reasoners. In *Proceedings of the 2008 International Workshop on Description Logics (DL2008), CEUR-WS*, pages 701–706, 2008.

[KC04] Graham Klyne and Jeremy J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. Technical report, `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`, 2004.

[LSK08] G. Lukacsy, P. Szeredi, and B. Kadar. Prolog based description logic reasoning. In *Proceedings of the 24th International*

Conference on Logic Programming (ICLP'08), pages 455–469, Heidelberg,Germany, 2008. Springer, LNCS 5366.

[LTBS04] Loredana Laera, Valentina A. M. Tamma, Trevor J. M. Bench-Capon, and Giovanni Semeraro. SweetProlog: A System to Integrate Ontologies and Rules. In Grigoris Antoniou and Harold Boley, editors, *RuleML*, pages 188–193, Heidelberg, 2004. Springer, LNCS 3323.

[MGH+09] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. OWL 2 Web Ontology: Reasoning in OWL 2 RL and RDF Graphs using Rules. Technical report, `http://www.w3.org/TR/owl2-profiles/#Reasoning_in_OWL_2_RL_and_RDF_Graphs_using_Rules`, 2009.

[MPG07] S. Munoz, J. Pérez, and C. Gutierrez. Minimal deductive systems for RDF. In *Proceedings of the 4th European conference on The Semantic Web: Research and Applications*, page 53, Heidelberg, Germany, 2007. Springer, LNCS 4519.

[MPSG09] Boris Motik, Peter F. Patel-Schneider, and Bernardo Cuenca Grau. OWL 2 Web Ontology Language Direct Semantics. Technical report, `http://www.w3.org/TR/owl2-direct-semantics/`, 2009.

[Sch09] Michael Schneider. OWL 2 Web Ontology Language RDF-Based Semantics. Technical report, `http://www.w3.org/TR/owl2-rdf-based-semantics/`, 2009.

[SPG+07] Evren Sirin, Bijan Parsia, Bernardo C. Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, June 2007.

[tH04] Herman J. ter Horst. Extending the rdfs entailment lemma. In *International Semantic Web Conference*, pages 77–91. Springer, LNCS 3298, 2004.

[TH06] D. Tsarkov and I. Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, pages 292–297. Springer, LNAI 4130, 2006.

[VWM09] V. Vassiliadis, J. Wielemaker, and C. Mungall. Processing OWL2 ontologies using Thea: An application of logic programming. In *Proceedings of the 6th International Workshop on OWL: Experiences and Directions (OWLED 2009)*, 2009.

[WHV08] J. Wielemaker, Z. Huang, and L. Van Der Meij. SWI-Prolog and the web. *Theory and Practice of Logic Programming*, 8(03):363–392, 2008.