Ontology Querying and Reasoning with XQuery *

Jesús M. Almendros-Jiménez
Dpto. de Lenguajes y Computación
University of Almería
jalmen@ual.es

ABSTRACT

In this paper we investigate an extension of XQuery for querying and reasoning with OWL-style ontologies. The proposed extension adds new primitives (i.e. boolean operators) in XQuery for querying OWL-style triples in such a way that XQuery can be used as query language for OWL. We also study how to implement the cited extension of XQuery into logic programming.

Categories and Subject Descriptors

D.3 [**PROGRAMMING LANGUAGES**]: Language Classifications – Constraint and logic languages; I.2.4 [**ARTIFICIAL INTELLIGENCE**]: Knowledge Representation Formalisms and Methods – Representation languages

Keywords

XQuery, Logic Programming, OWL, Semantic Web

1. INTRODUCTION

Web Ontology Language (OWL) [39] is a proposal of the *W3C consortium* [1] for ontology modeling. OWL is syntactically layered on *Resource Description Framework (RDF)* [41], a more simple ontology language, whose underlying model is based on triples. The *RDF Schema (RDFS)* [40] is also an ontology language, enriching RDF with specific vocabularies for meta-data. 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, a well-defined model of property inheritance, and similar extensions [39].

*(This work has been partially supported by the EU (FEDER) and the Spanish MEC under grant TIN2005-09207-C03-02, and the Spanish MICINN under grant TIN2008-06622-C03-03).

[1]http://www.w3.org.

International Workshop on Programming Language Techniques for XML (PLAN-X 2009) January 24, 2009, Savannah, Georgia.

On the other hand, *XQuery* [43, 10] is a typed functional language devoted to express queries against XML documents. It contains *XPath 2.0* [42] as a sublanguage. *XPath 2.0* supports navigation, selection and extraction of fragments from XML documents. *XQuery* also includes expressions to construct new XML values and to join multiple documents.

In this paper we present an extension of *XQuery* for the querying of XML combined with OWL and RDF(S) reasoning/querying, and we study how to implement it in logic programming, in particular, in Prolog. Such extension and implementation can be summarized as follows:

(a) XQuery is extended for the traversal of OWL statements;

(b) XQuery is extended with built-in boolean operators for handling OWL statements.

(c) the implementation of XQuery is based on the encoding of XQuery in logic programming;

(d) the extension of XQuery has to encode OWL in logic programming to be combined with XQuery queries, and thus,

(e) the extended XQuery can combine the querying of XML documents together with the reasoning with RDF (S) / OWL.

In recent works [3, 4, 2], we have proposed a logic programming based implementation of the *XPath* and *XQuery* languages. Such implementation allows to encode *XPath* and *XQuery* queries in logic programming. With this aim, *XML* documents are translated into a logic program by means of facts and rules, and an *XPath*/*XQuery* query is executed by specializing the logic program representing the input XML document and generating one or more specific goals for the query. From the computed answers for the goals we are able to rebuild the output XML document.

In the case of RDF(S), a great effort has been made for defining query languages for RDF documents (see [6, 18] for surveys about this topic). The proposals mainly fall on extensions of *SQL*-style syntax for handling the *graph based RDF structure*. In this line the most representative languages are SquishQL [29], SPARQL [13] and RQL [24]. Moreover, there are some languages based on extensions of *XPath*, *XSLT* and *XQuery* languages. In this line the most representative languages are XQuery for RDF (the Syntactic Web Approach) [31], RDF Twig [44], RDFPath [35],

RDFT [11] and XsRQL [25]. Finally, some of them are logic-based languages, therefore *rule based languages*. This is the case of TRIPLE [33], N3QL [8] and XCerpt [32, 15]. They have their own syntax similar to *deductive logic languages*. *XQuery* can be adapted to the handling of *RDF* documents by means of some kind of *serialization* of *RDF* documents in *XML*. Such serialization allows queries on RDF documents to be expressed by means of (extensions of) *XPath*. This serialization has been followed in some proposals [31, 44, 11, 35, 25], which study extensions of *XPath*, *XQuery* and *XSLT* for RDF.

In a recent proposal [1], we have studied how to define an extension of *XQuery* for querying RDF documents in which we have not to assume a given serialization of RDF, that is, our proposal handles RDF by means of a triple-based syntax. Such extension uses the **for** construction of *XQuery* for traversing RDF triples. In addition, our extension of *XQuery* is equipped with built-in boolean operators for RDF/RDFS properties like `rdf:type`, `rdfs:domain`, `rdfs:range`, `rdfs:subClassOf`, `rdfs:subPropertyOf`, and so on. Such built-in boolean operators can be used for reasoning about RDF, that is, for using the RDFS entailment relationship in the queries. Finally, we have studied a logic based implementation of this extension. In such implementation, logic rules are used for computing the RDFS entailment relationship from RDF(S) triples. Therefore the semantics of the built-in boolean operators is assumed to be defined by means of rules.

The aim of this paper is to introduce OWL reasoning in *XQuery* and to provide a logic programming based implementation of the extension. In order to extend our previous work for the handling of OWL we have to solve some troubles. The first one is how to handle OWL in an extension of *XQuery*. Fortunately, OWL can be expressed by means of RDF triples, and therefore we can handle OWL similarly to our extension to RDF. The second one is how to reason about OWL in *XQuery*. The solution will be also similar to the solution for RDF. We will study how to reason by means of logic rules with OWL and we will introduce new built-in boolean operators in *XQuery* for OWL reasoning which are defined by means of logic rules.

Therefore, in our proposal, XQuery supports the simultaneous querying of XML data by both its structure (as in classic XQuery) and by its associated meta-data given in form of OWL-style ontologies. This is an important problem for supporting data discovery tasks against collections of XML data. The problem of simultaneous querying of both data and meta-data has been intensely studied for other data models, but the topic is in its infancy for XML data. While query languages for XML + RDF do exist, their application in practice is restricted by the assumption of a particular encoding of the ontology hierarchy in XML. The extension proposed in this paper is independent of the XML encoding of ontologies, working directly on the conceptual RDF (and OWL) data model: triples.

Our approach will focus on the combination of OWL and logic programming. OWL is an ontology language based on the so-called *Description Logic (DL)* [9]. OWL is a language with different fragments named *OWL Full*, *OWL DL*, *OWL Lite* and *OWL Flight*, among others. Such fragments restrict the expressive power of OWL constructs in order to retain reasoning capabilities and decidability. OWL Full contains all the constructors of the OWL and allows the arbitrary combination of those constructors. OWL Full semantics is a extension of RDF semantics, however it yields to an undecidable reasoning language [22]. Therefore reasoning in OWL Full can be incomplete. OWL DL and OWL Lite are subsets of OWL Full in which some restrictions are considered. In particular, they are not properly semantically layered on top of RDFS. Therefore the RDF triple-based encoding of OWL DL and OWL Lite impose some restrictions about the RDF graphs. OWL Flight [12] is also based on OWL, but the semantics is grounded in logic programming rather than description logic. The reader can find more information about the topic in [38, 39, 19, 20, 30, 7, 36].

OWL has been recently combined and extended with logic rules in some works. The idea of such integration can be classified into two main lines of research.

The first line aims to study the intersection of OWL and logic programming, in other words, which fragment of OWL can be expressed in logic programming. In this research line, some authors [16, 38] have defined the so-called *Description Logic Programming*, which is the intersection of logic programming and description logic. Such intersection can be detected by encoding OWL into logic programming. With this aim, firstly, the corresponding fragment of OWL is represented by means of description logic, after such fragment of the description logic can be encoded into a fragment of *First Order Logic* (FOL); finally, the fragment of FOL can be encoded into logic programming.

Several fragments of OWL/DL can be encoded into logic programming, in particular, Volz [38] has encoded OWL subsets into *Datalog*, *Datalog(=)*, *Datalog(=,IC)* and *Prolog(=,IC)*; where "=" means "with equality", and "IC" means "with Integrity constraints". Some recent proposals have encoded description logic fragments into *disjunctive Datalog* [23], and into *Datalog(IC,≠,not)* (for OWL-Flight) [12], where "not" means "with negation". The reader can find more information about this topic in the proposals of [38] and [23].

The second line aims to extend a particular DL with logic rules on top of the ontology. This is the case of CARIN [28], $\mathcal{AL}$-log [14], TRIPLE [33], SWRL [21], SweetProlog [27], DR-Prolog [5] and F-OWL [46]. In such case, logic programming is used for reasoning with rules [26], but the particular DL is usually handled by means of external reasoners (for instance, *Racer* [17], FaCT++ [37], Pellet [34], KAON2 [23], among others).

We restrict our framework to the case of a simple kind of OWL ontology, which was proved can be encoded in Datalog in [38], and therefore it is possible to encode into Prolog. Therefore the handling of OWL in our framework is restricted to the same conditions as in [38] for the encoding into Datalog programs, and therefore decidability and complexity aspects of our approach are based on it.

Since the main aim of our approach is to exhibit the combination of querying of XML, RDF and OWL, the restriction to a simple kind of ontology makes our work easier. However, it does not affect substantially to the relevance of the approach once interesting examples can be handled in our query language. Our approach could be used in large-scale document repositories allowing document meta-data to be used in the generation of reports and aggregate documents. We believe that more complex ontologies which can be encoded into extensions of Datalog could be also integrated in our approach following the same ideas here presented. Such

Figure 1: Allowed Formulas

type $\mathcal{E}$	
A	atomic class
$C \sqcap D$	owl:intersectionOf
$\exists P.\{o\}$	owl:hasValue

type $\mathcal{L}$	
A	atomic class
$C \sqcap D$	owl:intersectionOf
$\exists P.\{o\}$	owl:hasValue
$C \sqcup D$	owl:unionOf
$\exists P.C$	owl:someValuesFrom

type $\mathcal{R}$	
A	atomic class
$C \sqcap D$	owl:intersectionOf
$\exists P.\{o\}$	owl:hasValue
$\forall P.C$	owl:allValuesFrom

Figure 2: An Example of Ontology

TBox

(1) Man $\sqsubseteq$ Person
(2) Woman $\sqsubseteq$ Person
(3) Person $\sqcap$ $\exists$author_of.Manuscript $\sqsubseteq$ Writer
(4) Paper $\sqcup$ Book $\sqsubseteq$ Manuscript
(5) Book $\sqcap$ $\exists$topic.{''XML''} $\sqsubseteq$ XMLbook
(6) Manuscript $\sqcap$ $\exists$reviewed_by.Person $\sqsubseteq$ Reviewed
(7) Manuscript $\sqsubseteq$ $\forall$rating.Score
(8) Manuscript $\sqsubseteq$ $\forall$topic.Topic
(9) author_of $\equiv$ writes
(10) average_rating $\sqsubseteq$ rating
(11) authored_by $\equiv$ author_of$^-$
(12) $\top \sqsubseteq \forall$ author_of.Manuscript
(13) $\top \sqsubseteq \forall$ author_of$^-$.Person
(14) $\top \sqsubseteq \forall$ reviewed_by.Person
(15) $\top \sqsubseteq \forall$ reviewed_by$^-$.Manuscript

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(''Abiteboul'',''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'')

extensions are considered as future work.

The structure of the paper is as follows. Section 2 will present the encoding of OWL documents into Prolog. Section 3 will present the extension of *XQuery* for OWL. Section 4 will present the encoding of the extension of XQuery and, finally, Section 5 will conclude and present future work.

2. ENCODING OWL IN LOGIC PROGRAMMING

In this section we show how to represent a certain class of DL-based ontologies in logic programming. This is the basis of our extension of *XQuery*. We restrict ourselves to the case of a kind of ontology expressible in Datalog [38]. Such kind of DL ontologies contains a **TBox** of axioms of the form:

$C \sqsubseteq D$	(rdfs:subClassOf)
$E \equiv F$	(owl:equivalentClass)
$P \sqsubseteq Q$	(rdfs:subPropertyOf)
$P \equiv Q$	(owl:equivalentProperty)
$P \equiv Q^-$	(owl:inverseOf)
$P \equiv P^-$	(owl:SymmetricProperty)
$P^+ \sqsubseteq P$	(owl:TransitiveProperty)
$\top \sqsubseteq \forall P^-.D$	(rdfs:domain)
$\top \sqsubseteq \forall P.D$	(rdfs:range)

where E, F are class descriptions of type $\mathcal{E}$ (see Figure 1), C is a class description of left hand side type (type $\mathcal{L}$, see Figure 1), and D is a class description of right-hand side type (type $\mathcal{R}$, see Figure 1), and P, Q are properties. In addition, the **ABox** contains axioms of the form:

$P(a, b)$	(property fillers)
$D(a)$	(individual assertion)

where P is a property, D is a class description of type $\mathcal{R}$, and a, b are individuals. Basically, the proposed subset of DL restricts occurrences of class descriptions in right and left hand sides of subclass and class equivalence axioms, and in individual assertions. Such restriction is required to be encoded by means of Datalog rules. Roughly speaking, the universal quantification is only allowed in the right hand side of DL formulas, which corresponds in the translation to the occurrences of the same quantifier in left hand side (i.e. head) of Horn clauses. The same can be said for union formulas which are required to appear in left hand sides which corresponds with the definition of two Horn clauses in the translation. In the case of existential quantifiers they are forbidden in the right hand side of DL formulas because they cannot be translated into left hand sides of Horn clauses. Let us see an example of a such DL ontology (see Figure 2).

The ontology of Figure 2 describes in the **TBox** metadata in which the elements of `Person` are elements of `Man` or elements of `Woman` (cases (1) and (2)); and the elements of `Manuscript` are either elements of `Paper` or elements of `Book` (case (4)). 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 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

Figure 3: Triple-based encoding of DL in FOL

$fol^t(C \sqsubseteq D) = \forall x.fol_x^t(C) \rightarrow fol_x^t(D)$	$fol_x^t(A) = triple(x, rdf:type, A)$
$fol^t(E \equiv F) = \forall x.fol_x^t(E) \leftrightarrow fol_x^t(F)$	$fol_x^t(C \sqcap D) = fol_x^t(C) \wedge fol_x^t(D)$
$fol^t(P \sqsubseteq Q) = \forall x, y.triple(x, p, y) \rightarrow triple(x, q, y)$	$fol_x^t(C \sqcup D) = fol_x^t(C) \vee fol_x^t(D)$
$fol^t(P \equiv Q) = \forall x, y.triple(x, p, y) \leftrightarrow triple(x, q, y)$	$fol_x^t(\exists P.C) = \exists y.triple(x, p, y) \wedge fol_y^t(C)$
$fol^t(P \equiv Q^-) = \forall x, y.triple(x, p, y) \leftrightarrow triple(y, q, x)$	$fol_x^t(\forall P.C) = \forall y.triple(x, p, y) \rightarrow fol_y^t(C)$
$fol^t(P \equiv P^-) = \forall x, y.triple(x, p, y) \leftrightarrow triple(y, p, x)$	$fol_x^t(\forall P^-.C) = \forall y.triple(y, p, x) \rightarrow fol_y^t(C)$
$fol^t(P^+ \sqsubseteq P) = \forall x, y, z.triple(x, p, y) \wedge triple(y, p, z) \rightarrow triple(x, p, z)$	$fol_x^t(\exists P.\{o\}) = \exists y.triple(x, p, y) \wedge y = o$
$fol^t(\top \sqsubseteq \forall P.C) = \forall x.fol_x^t(\forall P.C)$	
$fol^t(\top \sqsubseteq \forall P^-.C) = \forall x.fol_x^t(\forall P^-.C)$	

are described by means of a topic.

Now, we would like to show the encoding of such fragment of DL into logic programming. The encoding consists of two elements: a generic encoding for OWL reasoning, and an encoding for ontology instances.

2.1 Ontology Instance Encoding

The encoding uses Prolog facts for a predicate called `triple`. There is one fact for each element of an ontology instance. The encoding of DL formulas is as follows. Classes and properties are represented by means of Prolog atoms. Quantifiers are represented by means of Prolog terms, that is, $\forall P.C$ and $\exists P.C$ are represented by means of Prolog term `forall(p, c)` and `exists(p, c)`. Unions (i.e. $C \sqcup D$) and intersections (i.e. $C \sqcap D$) are also represented as `union(c, d)` and `inter(c, d)`, respectively. Inverse and transitivity properties (i.e. P^- and P^+) are represented as Prolog terms: `inv(P)` and `trans(P)`. Finally OWL relationships: `rdfs : subClassOf`, etc are represented as atoms in Prolog. Formally, the encoding is as follows.

In the case of the **TBox**: `en(C ⊑ D) = triple(en(C), rdfs : subClassOf, en(D))`; `en(E ≡ F) = triple(en(E), owl : equivalentClass, en(F))`; `en(P ⊑ Q) = triple(en(P), rdfs : subPropertyOf, en(D))`; and `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)` represent 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 $\top$ is encoded as `en(⊤) = thing`. In addition: `en(P+) = trans(en(P))`; `en(P−) = inv(en(P))`; `en(∀P.C) = forall(en(P), en(C))`; `en(∃P.C) = exists(en(P), en(C))`; `en(∃P.{o}) = exists(en(P), en(o))`; `en(C ⊓ D) = inter(en(C), en(D))`; and `en(C ⊔ D) = union(en(C), en(D))`.

Finally, the elements of the **ABox** are also encoded as Prolog facts relating pairs of individuals by means of properties, and defining memberships to classes: `en(P(a, b)) = triple(a, en(P), b)` and `en(D(a)) = triple(a, rdf : type, D)`. In the running example we will have:

```
triple(man,rdfs:subClassOf,person).
triple(woman,rdfs:subClassOf,person).
triple(inter(person,exists(author_of,manuscript)),
rdfs:subClassOf,writer).
triple(union(paper,book),rdfs:subClassOf,manuscript).
triple(inter(book,exists(topic,"XML")),rdfs:subClassOf,xmlbook).
triple(inter(manuscript,exists(reviewed_by,person)),
rdfs:subClassOf,reviewed).
triple(manuscript,rdfs:subClassOf,forall(rating,score)).
triple(manuscript,rdfs:subClassOf,forall(topic,topic)).
triple(author_of,owl:equivalentProperty,writes).
triple(authored_by,owl:equivalentProperty,inv(author_of)).
triple(average_rating,rdfs:subPropertyOf,rating).
triple(thing,rdfs:subClassOf, forall(author_of, manuscript)).
triple(thing,rdfs:subClassOf, forall(inv(author_of), person)).
triple(thing, rdfs:subClassOf, forall(reviewed_by,person)).
triple(thing, rdfs:subClassOf, forall(inv(reviewed_by),manuscript)).
triple("Abiteboul",rdf:type,man).
triple("Buneman",rdf:type,man).
triple("Suciu",rdf:type,man).
triple("Data on the Web",rdf:type,book).
triple("XML in Scottland",rdf:type,book).
triple("Growing XQuery",rdf:type,paper).
triple("Anonymous",rdf:type,person).
triple("Abiteboul",author_of,"Data on the Web").
triple("Data on the Web",authored_by,"Buneman").
triple("Suciu",author_of,"Data on the Web").
triple("Abiteboul",author_of,"XML in Scottland").
triple("Simeon",writes,"Growing XQuery").
triple("Data on the Web",reviewed_by,"Anonymous").
triple("Growing XQuery",reviewed_by,"Almendros").
triple("Data on the Web",average_rating,"good").
triple("XML in Scottland",rating,"excellent").
triple("Growing XQuery",rating,"good").
triple("Data on the Web",topic,"XML").
triple("Data on the Web",topic,"Web").
triple("XML in Scottland",topic,"XML").
```

2.2 Encoding for OWL reasoning

Now, the second element of the encoding consists of Prolog rules defining how to reason about OWL properties. Such rules express in a abstract and generic way the semantic information deduced from a given ontology instance. Such rules infer new relationships between the data in the form of triples. For instance, new triples for `rdf:type` are defined from the `rdfs:subClassOf` and `owl:equivalentClass` relationships. In addition, new triples are defined for properties. In particular, the `rdfs:subPropertyOf` and `owl:equivalentProperty` relationships induce new relationships. In the case of equivalence, the `owl:inverseOf` and `owl:TransitiveProperty`, encoded as Prolog terms built from `inv` and `trans`, induce also new triples. In order to define the encoding, we have to follow the encoding of DL into FOL, which is described in Figure 3. Such encoding is based on a representation by means of triples of the DL formulas, and in which classes and properties are considered as constants in FOL.

2.2.1 *Encoding Class Equivalence*

Equivalence formulas are encoded as follows. $A \equiv B$ is encoded as:

```
triple(X,rdf:type,A):-triple(X,rdf:type,B),
                      triple(A,owl:equivalentClass,B).
triple(X,rdf:type,B):-triple(X,rdf:type,A),
                      triple(B,owl:equivalentClass,A).
```

in which each triple of the form: `triple(A, owl : equiva-`

lentClass, B) induces a new triple from each triple of the form: triple(X, rdf : type, B) or triple(X, rdf : type, A). The case $A \equiv B \sqcap C$ is encoded as follows:

```
triple(X, rdf : type, A) : −triple(X, rdf : type, B),
                triple(X, rdf : type, C),
                 triple(A, owl : equivalentClass, inter(B, C)).
triple(X, rdf : type, B) : −triple(X, rdf : type, A),
                triple(A, owl : equivalentClass, inter(B, C)).
triple(X, rdf : type, C) : −triple(X, rdf : type, A),
                triple(A, owl : equivalentClass, inter(B, C)).
```

in which the occurrence of a triple of the form triple(A, owl : equivalentClass, inter(B, C)) in a fact of an ontology instance induce new triples. Similarly, in the case of $A \equiv \exists P.\{o\}$, the encoding is:

```
triple(X, rdf : type, A) : −triple(X, P, O),
                triple(A, owl : equivalentClass, exists(P, O)).
triple(X, P, O) : −triple(X, rdf : type, A),
                triple(A, owl : equivalentClass, exists(P, O)).
```

Let us remark that rules for equivalence relationships lead to loops in a Prolog interpreter. However, this is not a serious problem by modifying the interpreter with memorization of OWL triples.

2.2.2 Encoding Class Inclusion

Inclusion formulas are encoded as follows. The case $C \sqsubseteq D$ is as follows:

```
triple(X,rdf:type,D):-triple(X,rdf:type,C),
                triple(C,rdfs:subClassOf,D).
```

which encodes the meaning of the subclass relationship. Let us remark that inclusion can also lead to loops in a Prolog interpreter when cyclic definitions are allowed, but we can also memorize triples to solve this problem.

The case of inclusion relationships with existential (i.e. $\exists P.C$) and universal quantifiers (i.e. $\forall P.C$) is as follows. Let us remark that universal quantifiers only occur in right hand sides, and therefore $A \sqsubseteq \forall P.C$ is encoded as:

```
triple(Y, rdf : type, C) : −triple(X, rdf : type, A),
        triple(X, P, Y), triple(A, rdfs : subClassOf, forall(P, C)).
```

The case $\exists P.C \sqsubseteq A$ is encoded as:

```
triple(X, rdf : type, A) : −triple(Y, rdf : type, C),
        triple(X, P, Y), triple(exists(P, C), rdfs : subClassOf, A).
```

Analogously, the cases $A \sqsubseteq \exists P.\{o\}$, $\exists P.\{o\} \sqsubseteq A$. The case $B \sqcup C \sqsubseteq A$ is encoded as:

```
triple(X,rdf:type,A):-triple(X,rdf:type,B),
                 triple(union(B,C),rdfs:subClassOf,A).
triple(X,rdf:type,A):-triple(X,rdf:type,C),
                triple(union(B,C),rdfs:subClassOf,A).
```

Analogously for intersections, $A \sqsubseteq B \sqcap C$ is encoded as:

```
triple(X,rdf:type,B):-triple(X,rdf:type,A),
                triple(A,rdfs:subClassOf,inter(B,C)).
triple(X,rdf:type,C):-triple(X,rdf:type,A),
                triple(A,rdfs:subClassOf,inter(B,C)).
```

and similarly $B \sqcap C \sqsubseteq A$:

```
triple(X,rdf:type,A):-triple(X,rdf:type,B),
                triple(X,rdf:type,C),
                triple(inter(B,C),rdfs:subClassOf,A).
```

2.2.3 Encoding Property Equivalence and Inclusion

In the case of equivalence $P1 \equiv P2$ for properties we would have:

```
triple(X,P1,Y):-triple(X,P2,Y),
        triple(P1,owl:equivalentProperty,P2),atomic(P2).
triple(X,P1,Y):-triple(X,P2,Y),
        triple(P2,owl:equivalentProperty,P1),atomic(P1).
```

where atom(P1) and atom(P2) are included in order to detect the occurrences of cases P, inv(P), trans(P). In the case of inclusion, we have:

```
triple(X,P2,Y):-triple(X,P1,Y),
        triple(P1,rdfs:subPropertyOf,P2),atomic(P2).
```

and we have special rules for handling inverse $P1 \equiv P2^{-}$:

```
triple(Y,P2,X):-triple(X,P1,Y),
        triple(P1,owl:equivalentProperty,inv(P2)).
```

and transitivity $P^{+} \sqsubseteq P$:

```
triple(X,P2,Y):-triple(X,P1,Y),
        triple(P1,owl:equivalentProperty,trans(P2)).
triple(X,P2,Z):-triple(X,P1,Y),triple(Y,P1,Z),
        triple(P1,owl:equivalentProperty,trans(P2)).
```

which compute the transitive closure of a relationship.

2.2.4 DL formulas encoding

Finally, we have to add rules for DL formulas encoding as follows. In the case of $C \sqcap D$ we have:

```
triple(X,rdf:type,inter(C,D)):-triple(X,rdf:type,C),
        triple(X,rdf:type,D).
triple(X,rdf:type,C):-triple(X,rdf:type,inter(C,D)).
triple(X,rdf:type,D):-triple(X,rdf:type,inter(C,D)).
```

which induce new triples from the given ontology instance. Such triples could be already included in the ontology instance (as facts) and they could not introduce new information in some cases. In the case of $C \sqcup D$ we have:

```
triple(X,rdf:type,union(C,D)):-triple(X,rdf:type,C).
triple(X,rdf:type,union(C,D)):-triple(X,rdf:type,D).
```

Finally, for existential quantifiers we would have the following rules. In the case of $\exists P.C$:

```
triple(X,rdf:type,exists(P,C)):-triple(X,P,Y),
                triple(Y,rdf:type,C).
```

and similarly $\exists P.\{o\}$:

```
triple(X,rdf:type,exists(P,O)):-triple(X,P,O).
```

Finally, there is a fact triple(thing, rdf : type, A) (where A is a variable) required for the encoding of range and domain statements. We have exhibited the rules of the encoding of rdf:type. Some other rules can be considered for reasoning about other OWL properties.

2.2.5 Prolog as Inference Engine

The proposed encoding allows to use Prolog as inference engine for OWL. For instance, the triple triple(‘‘Data on the Web’’, rdf:type, reviewed) is deduced from the following facts:

```
triple(‘‘Data on the Web’’,rdf:type,book).
triple(book,rdfs:subClassOf,manuscript).
triple(‘‘Data on the Web’’,reviewed_by,‘‘Anonymous’’).
triple(‘‘Anonymous’’,rdf:type,person).
triple(inter(manuscript,exists(reviewed_by,person)),
                        rdfs:subClassOf,reviewed).
```

Figure 4: Core XQuery

xquery:= **namespace** *name* **:** *resource* **in** *xquery*
| *dexpr* | *< tag att = vexpr, . . . , att = vexpr >'{'xquery, . . . , xquery'}'< /tag >* | *flwr* | *value.*
dexpr:= **document***(doc) '/' expr.*
rdfdoc := **rdfdocument***(doc).*
owldoc := **owldocument***(doc).*
tripledoc:= rdfdoc | *owldoc.*
flwr:= **for** *$var* **in** *vexpr [***where** *constraint]* **return** *xqvar*
| **for** *($var,$var,$var)* **in** *tripledoc [***where** *constraint]* **return** *xqvar*
| **let** *$var* **:=** *vexpr [***where** *constraint]* **return** *xqvar.*
xqvar:= vexpr | < tag att = vexpr, . . . , att = vexpr >'{'xqvar, . . . , xqvar'}'< /tag > | flwr | value.
vexpr:= $var | $var '/' expr | dexpr | value.
expr:= text() | tag | tag[expr] | '/' expr.
*constraint := Op(vexpr, . . . , vexpr) | constraint '***or***' constraint | constraint '***and***' constraint.*

and the following set of rules:

```
triple(X,rdf:type,Y):-triple(X,rdf:type,Z),
                      triple(Z,rdfs:subClassOf,Y).
triple(X,rdf:type,exists(P,C)):-triple(X,P,Y),
               triple(Y,rdf:type,C).
triple(X,rdf:type,A):-triple(X,rdf:type,B),
                      triple(X,rdf:type,C),
                      triple(inter(B,C),rdfs:subClassOf,A).
```

Also the triple `triple(''Buneman'',author_of,''Data on the Web'')` is deduced from the following facts:

```
triple(''Data on the Web'',authored_by,''Buneman'').
triple(authored_by,owl:equivalentProperty,inv(author_of)).
```

and the following rule:

```
triple(Y,P2,X):-triple(X,P1,Y),
       triple(P1,owl:equivalentProperty,inv(P2)).
```

3. EXTENDED XQUERY

Now, we would like to show how to query an OWL ontology by means of *XQuery*. The grammar of the extended *XQuery* for querying XML, RDF and OWL in our framework is described in Figure 4, where "name **:** resource" assigns name spaces to URL resources; "value" can be URLs / URIs, name spaces, strings, numbers or XML documents; tag's are XML labels; att's are attribute names; doc's are URLs; and finally, *Op*'s can be selected from the usual binary operators: <=, >=, <, >, =, =/=, and OWL/RDF(S) built-in boolean operators.

Basically, a simple *XQuery* language has been extended as follows:

- The **namespace** statement has been added allowing the declaration of URIs taken from other resources;
- A new **for** expression has been added for traversing triples from a RDF document whose location is specified by means of **rdfdocument** primitive; analogously, a new **for** expression for traversing triples from an OWL document whose location is specified by means of a **owldocument** primitive.
- In addition, the **where** construction includes boolean conditions of the form $Op(vexpr, \ldots, vexpr)$ which can be used for testing RDF(S)/OWL properties. The predicate *Op* can be one of `rdf:type`, `rdfs:subClassOf`, `owl:equivalentClass`, etc.

The above *XQuery* is a typed language in which there are two kinds of variables: those variables used in *XPath* expressions, and those used in RDF(S)/OWL triples. However they can be compared by means of boolean expressions, and they can be used together for the construction of the answer.

We have considered a subset of the *XQuery* language in which some other built-in constructions for *XPath* can be added, and also it can be enriched with other *XQuery* constructions, following the W3C recommendations [43]. However, with this small extension of *XQuery* we are able to express interesting queries in XML, RDF and OWL.

Now, we will focus on show how we can express queries and how we can reason with OWL. We would like to show an example of query in order to give us an idea about how the proposed extension of *XQuery* is suitable for OWL querying and reasoning. The query we like to show is *"Retrieve the authors of books"*. It can be expressed in our proposed extension of *XQuery* as follows:

```
< list > {
for  ($Author,$Property,$Book) in owldocument(''ex.owl'')
return
for  ($Book2,$Property2,$Type) in owldocument(''ex.owl'')
where $Book=$Book2 and rdfs:subPropertyOf($Property,author_of)
and $Property2=rdf:typeOf and rdfs:subClassOf($Type,book)
return
<author>{ $Author } </author >
} </> list >
```

In the example we can see the following elements.

- OWL triples are traversed by means of the **for** construct of *XQuery*. Each triple is described by means of three variables (prefixed with '$' as usual in *XQuery*). Given that we have to query two properties, that is, `rdf:typeOf` and `author_of`, we have to combine two **for** expressions.
- The **where** expression has to check whether each pair of triples corresponds with the same book by means of the condition `$Book=$Book2`.
- In addition, the first property, that is, `$Property`, has to be a subproperty of `author_of`, and the second property has to be `rdf:typeOf`. Let us remark that we could write `$Property=author_of` instead of `rdfs:subPropertyOf($Property,author_of)` but in such a case only triples "`author_of`" would be considered, and not subproperties of "`author_of`".
- The type of the book should be "`Book`", an it is checked by means of `rdfs:subClassOf($Type,book)`.
- Finally, the output of the query is shown by means of

Figure 5: A Subset of the Core XQuery

xquery:= **namespace** *name* **:** *resource* **in** *xquery*
| *< tag att = vexpr, . . . , att = vexpr >'{'xquery, . . . , xquery'}'< /tag >* | *flwr* | *value.*
owldoc := owldocument(doc).
flwr:= **for** *($var,$var,$var)* **in** *owldoc [***where** *constraint]* **return** *xqvar.*
xqvar:= vexpr | < tag att = vexpr, . . . , att = vexpr >'{'xqvar, . . . , xqvar'}'< /tag > | flwr | value.
vexpr:= $var | value.
*constraint := Op(vexpr, . . . , vexpr) | constraint '***or***' constraint | constraint '***and***' constraint.*

Figure 6: Encoding of XML documents

Rules (Schema):

```
books(bookstype(Book, []), NBooks,1,Doc) :-
      book(Book, [NBook|NBooks],2,Doc).
book(booktype(Author, Title, Review,
      [year=Year]),NBook ,2,Doc) :-
      author(Author, [NAu|NBook],3,Doc),
      title(Title, [NTitle|NBook],3,Doc),
      review(Review, [NRe|NBook],3,Doc),
      year(Year, NBook,3,Doc).
review(reviewtype(Un,Em,[]),NReview,3,Doc):-
    unlabeled(Un,[NUn|NReview],4,Doc),
    em(Em,[NEm|NReview],4,Doc).
review(reviewtype(Em,[]),NReview,3,Doc):-
    em(Em,[NEm|NReview],5,Doc).
em(emtype(Unlabeled,Em,[]),NEms,5,Doc) :-
    unlabeled(Unlabeled,[NUn|NEms],6,Doc),
    em(Em, [NEm|NEms],6,Doc).
```

Facts (Document):

```
year('2003', [1, 1], 3,"books.xml").
author('Abiteboul', [1, 1, 1], 3,"books.xml").
author('Buneman', [2,1, 1], 3,"books.xml").
author('Suciu', [3,1,1], 3,"books.xml").
title('Data on the Web', [4, 1, 1], 3,"books.xml").
unlabeled('A', [1, 5, 1, 1], 4,"books.xml").
em('fine', [2, 5, 1, 1], 4,"books.xml").
unlabeled('book.', [3, 5, 1, 1], 4,"books.xml").
year('2002', [2, 1], 3,"books.xml").
author('Buneman', [1, 2, 1], 3,"books.xml").
title('XML in Scotland', [2, 2, 1], 3,"books.xml").
unlabeled('The', [1, 1, 3, 2, 1], 6,"books.xml").
em('best', [2, 1, 3, 2, 1], 6,"books.xml").
unlabeled('ever!', [3, 1, 3, 2, 1], 6,"books.xml").
```

XML in which each element (i.e. the author) is labeled by means of "`author`".

In this case the answer would be:

```
<list>
<author>Abiteboul</author>
<author>Suciu</author>
<author>Buneman</author>
<author>Abiteboul</author>
</list>
```

4. ENCODING THE EXTENDED XQUERY

In this section, we would like to show how the extended XQuery language can be encoded in logic programming (in particular, in Prolog). Therefore, our proposed query language can be executed under Prolog.

In order to make the paper self-contained we will restrict to a fragment of the extended *XQuery* for querying and reasoning with OWL triples and for generating XML documents as output. The encoding of the whole extended *XQuery* has to combine in a uniform way the encoding of [3] for XML querying and the encoding of [1] for RDF querying and reasoning. The fragment to be encoded will consist in the grammar of the Figure 5, where *Op* can be one of the OWL built-in boolean operators. The reader can check that such fragment has been used in the example of the previous section. Now, a crucial point of the encoding is the encoding of XML documents in logic programming. Such encoding will allow to encode such fragment of *XQuery*, in particular, the output of the query. Next section will show an example of the encoding of XML documents. A formal and complete definition can be found in [4].

4.1 Encoding XML Documents

Let us consider the following document called "books.xml":

```
<books>
<book year="2003">
    <author>Abiteboul</author>
    <author>Buneman</author>
    <author>Suciu</author>
    <title>Data on the Web</title>
<review>A <em>fine</em> book.</review>
</book>
<book year="2002">
    <author>Buneman</author>
    <title>XML in Scotland</title>
    <review><em>The <em>best</em> ever!</em></review>
</book> </books>
```

Now, it can be represented by means of a logic program as in Figure 6. In our encoding we can distinguish the so-called *schema rules* which define the structure of the XML document, and *facts* which store the leaves of the XML tree (and therefore the values of the XML document). Each tag is translated into a predicate name: *books*, *book*, etc. Each predicate has four arguments, the first one, used for representing the XML document structure, is encapsulated into a function symbol with the same name as the tag adding the suffix *type*. Therefore, we have *bookstype*, *booktype*, etc. The second argument is used for numbering each node (a list of natural numbers identifying each node); the third argument of the predicates is used for numbering each type (a natural number identifying each type); and the last argument represents the document name. The key element of our translation is to be able to recover the original XML document from the set of rules and facts. The translation has the following peculiarities. In order to specify the order of an XML document in a fact based representation, each fact is numbered (from left to right and by levels in the XML tree). In addition, the hierarchical structure of the XML records is described by means of the identifier of each record: the length of the numbers of the children is larger than the number of the parent. The type number makes possible to map schema rules with facts.

4.2 Encoding of XQuery

Now, we will show how to encode the proposed extension of *XQuery* that uses the XML encoding for the building of the query answers.

Query 1: This query expresses *"Retrieve the authors of books"* in *XQuery* as follows:

```
< list > {
for  ($Author,$Property,$Book) in owldocument(''ex.owl'')
return
for  ($Book2,$Property2,$Type) in owldocument(''ex.owl'')
where $Book=$Book2 and rdfs:subPropertyOf($Property,author_of)
and $Property2=rdf:typeOf and rdfs:subClassOf($Type,book)
return
<author>{ $Author } </author >
} </> list >
```

Now, the encoding is as follows:

```
(1) list(listtype(Author,[]),NList,1,Doc):-
      author(Author,[NAuthor|NList],2,Doc).
(2) author(authortype(Author,[]),NAuthor,2,''result.xml''):-
      join(Author,NAuthor).
(3) join(Author,[NAuthor,[1]]):-
      triple(Author,Property,Book,NAuthor,''ex.owl''),
      triple(Book2,Property2,Type,_,''ex.owl''),
      eq(Book,Book2),
      triple(Property,rdfs:subPropertyOf,author_of,_,
     ''ex.owl''),
      eq(Property2,rdf:typeOf),
      triple(Type,rdfs:subClassOf,book,_,''ex.owl'').
```

The encoding takes into account the following elements.

- The **return** expression generates an XML document, and therefore there are some schema rules of the output document. In the previous example, this is the case of rule **(1)**, describing that the `list` label includes elements labeled as `author`.
- The rules **(2)** and **(3)** are the main rules of the encoding, in which the elements of the output document are computed by means of the so-called `join` predicate.
- The `join` predicate is the responsible of the encoding of the **for** and **where** constructs. Each **for** associated to an OWL triple is encoded by means of a call to the `triple` predicate with variables.
- Now, the **where** expression is encoded as follows. In the case of binary operators like "=", ">", ">=", etc, they are encoded by means of Prolog predicates, `eq`, `ge`, `geq`, etc. In the case of built-in boolean operators of the kind `rdf:typeOf`, `rdfs:subClassOf`, etc, a call to the `triple` predicate is achieved.
- Finally, we have to incorporate to the `triple` predicate two new arguments. The first new argument is a list of natural numbers identifying the triple. Each element of the **TBox** is identified by means of [1] ,[2], etc. The triples inferred from the **TBox** can be identified by appending the identifiers of the triples used for the inference. Therefore, in general, each triple can be identified as a list of natural numbers (for example $[1, 4, 5]$). Triple identifiers are required for representing the order of the output XML document. The second new argument is the name of the document storing the triple.

Let us remark that the `triple` predicate is defined by means of facts and rules (see Section 2) and therefore queries in *XQuery* assume the use of the inference in the presence of OWL boolean operators. Now, the Prolog goal `?-author(Author,Node,Type,Doc)` has the following answers:

```
Author=authortype("Abiteboul",[]),Node=[[15],[1]],
      Type=2,Doc=''result.xml''
Author=authortype("Suciu",[]),Node=[[17],[1]],
      Type=2,Doc=''result.xml''
Author=authortype("Buneman",[]),Node=[[16,6],[1]],
      Type=2,Doc=''result.xml''
Author=authortype("Abiteboul",[]),Node=[[18],[1]],
      Type=2,Doc=''result.xml''
```

and from the schema rule (rule (**1**)), and these computed answers, we can rebuild the output XML document:

```
<list>
<author>Abiteboul</author>
<author>Suciu</author>
<author>Buneman</author>
<author>Abiteboul</author>
</list>
```

Query 2: The second query we would like to show is *"Retrieve the books of topic XML"*.

Now, the properties to be checked in the query are "`topic`" and "`rdf:typeOf`". However, given that the ontology already includes the class "`XMLBook`" we can express the query as follows:

```
< list > {
for  ($Book,$Property,$Type) in owldocument(''ex.owl'')
where $Property=rdf:typeOf and rdfs:subClassOf($Type,XMLBook)
return
<book>{ $Book }</ book >
}
</> list >
```

In this case, the answer would be:

```
<list>
<book>Data on the Web</book>
<book>XML in Scottland</book>
</list>
```

Now, the encoding is as follows:

```
list(listtype(Book,[]),NL,1,Doc):-book(Book,[NB|NL],2,Doc).
book(booktype(Book,[]),NB,2,''result.xml''):-join(Book,NB).
join(Book,[NBook,[1]]):-
     triple(Book,Property,Type,NBook,''ex.owl''),
     eq(Property,rdf:typeOf),
     triple(Type,rdfs:subClassOf,xmlbook,_,''ex.owl").
```

and the goal is `?-book(Book,Node,Type,Doc)`.

Query 3: The third query we would like to show is *"Retrieve the reviewed manuscripts and their writers"*. In this case, the query can be expressed in *XQuery* as follows:

```
< list > {
for ($Writer,$Property,$Manus) in owldocument(''ex.owl'')
for ($Manus2,$Property2,$Type) in owldocument(''ex.owl'')
where $Manus=$Manus2 and rdfs:subPropertyOf($Property,writes)
and $Property2=rdf:typeOf and rdfs:subClassOf($Type,reviewed)
return
<item>{
<manuscript> { $Manus } </ manuscript >
<writer>{ $Writer } </writer >
} </item>
}
</> list >
```

In this case, the properties to be checked are "`writes`" and "`rdf:typeOf`". The boolean operator `rdfs : subPropertyOf ($Property, writes)` is also true for triples in which the `$Property` is "`author_of`" given that both are equivalent properties in the given ontology. A more concise way to express the previous query is:

```
< list > {
for ($Writer,$Property,$Manuscript) in
owldocument(''ex.owl'')
where rdfs:subPropertyOf($Property,writes) and
rdf:type(Manuscript,reviewed)
return
<manuscript> { $Manuscript } </ manuscript >
<writer>{ $Writer } </writer >
}
</> list >
```

using the built-in boolean operator `rdf:type` whose semantics takes into account the inclusion and equivalence relationships of the input ontology. Now, the goal is $?-$ `item(Item, Node, Type, Doc)` and the encoding (in the first

case) is as follows:

```
list(listtype(Item,[]),NL,1,Doc):- item(Item,[NItem|N],2,Doc).
item(itemtype(manuscripttype(Manus,[]),writertype(Writer,
     []),[]),NItem,2,‘‘result.xml’’):-
     join(Manus,Writer,NItem).
join(Manuscript,Writer,[NM,[1]]):-
     triple(Writer,Property,Manuscript,NM,‘‘ex.owl’’),
     triple(Manuscript2,Property2,Type,_,‘‘ex.owl’’),
     eq(Manuscript,Manuscript2),
     triple(Property,rdfs:subPropertyOf,writes,_,
    ‘‘ex.owl’’),
     eq(Property2,rdf:typeOf),
     triple(Type,rdfs:subClassOf,reviewed,_,‘‘ex.owl’’).
```

Query 4: Finally, we would like to show the query *"Retrieve the equivalent properties of the ontology"*, in which we can extract from the input ontology some properties and to represent the output of the query as an ontology:

```
< owl:Ontology > {
for ($Object1,$Property,$Object2) in owldocument(‘‘ex.owl’’)
where $Property=owl:equivalentProperty
return
<owl:ObjectProperty rdf:about=$Object1/>
<owl:equivalentProperty rdf:resource=$Object2 />
</owl:ObjectProperty>
}
</ owl:Ontology >
```

Now, the goal is $? - \texttt{owlObjectProperty}(\texttt{OwlObjectProperty}, \texttt{Node}, \texttt{Type}, \texttt{Doc})$, and it is encoded as follows:

```
owlOntology(owlOntologytype(OwlObjectProp,[]),Nowl,1,Doc):-
      owlObjectProperty(OwlObjectProp,[Nowlp|Nowl],2,Doc).
owlObjectProperty(owlObjectPropertytype(equivalent
     Propertytype(‘‘’’,[rdfresource=Object2]),[rdfabout=Object1]),
     Nowlp,2,‘‘result.xml’’):-
        join(Object1,Object2,Nowlp).
join(OwlOntology,[NTriple,[1]]):-
       triple(Object1,Property,Object2,NTriple,‘‘ex.owl’’),
       eq(Property,owl:equivalentProperty).
```

5. CONCLUSIONS AND FUTURE WORK

In this paper we have studied an extension of *XQuery* for the querying and reasoning with OWL style ontologies. Such extension combines RDF(S)/OWL and XML documents as input/output documents. By means of built-in boolean operators *XQuery* can be equipped with inference mechanism for OWL properties. We have also studied how to implement/encode such language in logic programming. The proposed encoding can be generalized and achieved in an automatic way. We are now developing a formal translation of the extended *XQuery* into logic programming in order to be implemented. We also would like to develop a prototype of our language using the SWI-Prolog platform and the RDF/OWL library. We can take advantage from the RDF storing and retrieval of SWI-Prolog [45] for the implementation of the proposed extension of *XQuery* into *Prolog*.

6. REFERENCES

[1] J. M. Almendros-Jiménez. An RDF Query Language based on Logic Programming. *Electronic Notes in Theoretical Computer Science*, 200(3), 2008.

[2] 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.

[3] J. M. Almendros-Jiménez, A. Becerra-Terón, and Francisco J. Enciso-Baños. Integrating XQuery and Logic Programming. In *Proceedings of the INAP/WLP'07*, Hieldelberg, Germany, 2007. Springer LNAI, To appear.

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

[5] Grigoris Antoniou and Antonis Bikakis. DR-Prolog: A System for Defeasible Reasoning with Rules and Ontologies on the Semantic Web. *IEEE Trans. on Knowl. and Data Eng.*, 19(2):233–245, 2007.

[6] James Bailey, Franois Bry, Tim Furche, and Sebastian Schaffert. Web and Semantic Web Query Languages: A Survey. In *Proc. of Reasoning Web, First International Summer School*, pages 35–133, Heidelberg, Germany, 2005. Springer LNCS 3564.

[7] Sean K. Bechhofer and Jeremy J. Carroll. Parsing OWL DL: Trees or Triples? In *WWW '04: Proceedings of the 13th International Conference on World Wide Web*, pages 266–275, New York, NY, USA, 2004. ACM Press.

[8] Tim Berners-Lee. N3QL-RDF Data Query Language. Technical report, Online only, 2004.

[9] Alex Borgida. On the relative expressiveness of Description Logics and Predicate Logics. *Artificial Intelligence*, 82(1-2):353–367, 1996.

[10] D. Chamberlin, Denise Draper, Mary Fernández, Michael Kay, Jonathan Robie, Michael Rys, Jerome Simeon, Jim Tivy, and Philip Wadler. *XQuery from the Experts*. Addison Wesley, Boston, USA, 2004.

[11] I. Davis. RDF Template Language 1.0. Technical report, Online only, September 2003.

[12] Jos de Bruijn, Rubén Lara, Axel Polleres, and Dieter Fensel. OWL DL vs. OWL Flight: conceptual modeling and reasoning for the semantic Web. In *WWW '05: Proceedings of the 14th International Conference on World Wide Web*, pages 623–632, New York, NY, USA, 2005. ACM Press.

[13] Cristian Pérez de Laborda and Stefan Conrad. Bringing Relational Data into the Semantic Web using SPARQL and Relational OWL. In *Procs. of ICDEW'06*, page 55, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[14] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. AL-log: integrating Datalog and description logics. *J. of Intelligent and Cooperative Information Systems*, 10:227–252, 1998.

[15] Tim Furche, François Bry, and Oliver Bolzer. Marriages of Convenience: Triples and Graphs, RDF and XML in Web Querying. In *Proceedings of Third Workshop on Principles and Practice of Semantic Web Reasoning*, pages 72–84, Heidelberg,Germany, 2005. REWERSE, Springer LNCS 3703.

[16] 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, USA, 2003. ACM Press.

[17] V. Haarslev and R. Möller. Racer: An OWL Reasoning Agent for the Semantic Web. In *Proceedings of the International Workshop on Applications, Products and Services of Web-based*

Support Systems, pages 91–95, 2003.

[18] Peter Haase, Jeen Broekstra, Andreas Eberhart, and Raphael Volz. A Comparison of RDF query languages. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *Proceedings of the Third International Semantic Web Conference*, pages 502–517, Heidelberg, Germany, November 2004. Springer LNCS 3298.

[19] Jeff Heflin and Zhengxiang Pan. A Model Theoretic Semantics for Ontology Versioning. In *International Semantic Web Conference*, pages 62–76, Hiedelberg, Germany, 2004. Springer LNCS 3298.

[20] Ian Horrocks and Peter F. Patel-Schneider. Reducing OWL Entailment to Description Logic Satisfability. In *Description Logics*, 2003.

[21] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission, 21 May 2004. Available at http://www.w3.org/Submission/SWRL/.

[22] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: the making of a Web Ontology Language. *J. Web Sem.*, 1(1):7–26, 2003.

[23] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reasoning in Description Logics by a Reduction to Disjunctive Datalog. *J. Autom. Reasoning*, 39(3):351–384, 2007.

[24] Gregory Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, and Michel Scholl. RQL: a declarative query language for RDF. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 592–603, New York, NY, USA, 2002. ACM Press.

[25] H. Katz. XsRQL: an XQuery-style Query Language for RDF. Technical report, Online only, 2004.

[26] Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. Description Logic Rules. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI-08)*. IOS Press, 2008.

[27] 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,Germany, 2004. Springer LNCS 3323.

[28] Alon Y. Levy and Marie-Christine Rousset. Combining Horn rules and Description Logics in CARIN. *Artif. Intell.*, 104(1-2):165–209, 1998.

[29] Libby Miller, Andy Seaborne, and Alberto Reggiori. Three Implementations of SquishQL, a Simple RDF Query Language. In *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web*, pages 423–435, Heidelberg,Germany, 2002. Springer.

[30] Jeff Z. Pan and Ian Horrocks. RDFS(FA): Connecting RDF(S) and OWL DL. *IEEE Trans. Knowl. Data Eng.*, 19(2):192–206, 2007.

[31] Jonathan Robie, Lars Marius Garshol, Steve Newcomb, Michel Biezunski, Matthew Fuchs, Libby Miller, Dan Brickley, Vassilis Christophides, and Gregorius Karvounarakis. The Syntactic Web: Syntax and Semantics on the Web. *Markup Languages: Theory & Practice*, 4(3):411–440, 2002.

[32] S. Schaffert and F. Bry. A Gentle Introduction to Xcerpt, a Rule-based Query and Transformation Language for XML. In *Proc. of International Workshop on Rule Markup Languages for Business Rules on the Semantic Web*, page 22 pages, Aachen, Germany, 2002. CEUR Workshop Proceedings 60.

[33] Michael Sintek and Stefan Decker. TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web. In *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web*, pages 364–378, Heidelberg, Germany, 2002. Springer.

[34] 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.

[35] Adam Souzis. RxPath: a mapping of RDF to the XPath Data Model. In *Extreme Markup Languages*, 2006.

[36] Herman J. ter Horst. Combining RDF and Part of OWL with Rules: Semantics, Decidability, Complexity. In *International Semantic Web Conference*, pages 668–684, 2005.

[37] 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, Heidelberg,Germany, 2006. Springer LNAI 4130.

[38] Raphael Volz. *Web Ontology Reasoning with Logic Databases*. PhD thesis, Universität Fridericiana zu Karlsruhe, 2004.

[39] W3C. OWL Ontology Web Language. Technical report, www.w3.org, 2004.

[40] W3C. RDF Vocabulary Description Language 1.0: RDF Schema. Technical report, www.w3.org, 2004.

[41] W3C. Resource Description Framework (RDF). Technical report, www.w3.org, 2004.

[42] W3C. XML Path Language (XPath) 2.0. Technical report, www.w3.org, 2007.

[43] W3C. XML Query Working Group and XSL Working Group, XQuery 1.0: An XML Query Language. Technical report, www.w3.org, 2007.

[44] Norman Walsh. RDF Twig: Accessing RDF Graphs in XSLT. In *Proceedings of Extreme Markup Languages*, 2003.

[45] J. Wielemaker, G. Schreiber, and B. J. Wielinga. Prolog-Based Infrastructure for RDF: Scalability and Performance. In *International Semantic Web Conference*, pages 644–658, 2003.

[46] Youyong Zou, Tim Finin, and Harry Chen. F-OWL: an Inference Engine for the Semantic Web. In *Formal Approaches to Agent-Based Systems*, Heidelberg,Germany, 2004. Springer LNCS 3228.