Implementing OWL Lite in rule-based systems and recursion-enabled relational DBs

Raphael Volz
Forschungszentrum Informatik (FZI)
Institute AIFB
University of Karlsruhe, Karlsruhe, Germany
volz@fzi.de

October 4, 2002

"Let it be assumed that the states by virtue of which the soul possesses truth by way of affirmation or denial are five in number, i.e. art, scientific knowledge, practical wisdom, philosophic wisdom, intuitive reason; we do not include judgement and opinion because in these we may be mistaken." [Ari, idea 3 paragraph 1]

1 Introduction

The aim of this work is to develop arguments for the decision on the set of primitives, which should be available for the lower conformance layer (OWL^{Lite}) of the proposed Web Ontology language (OWL).

Several arguments were brought forward for having such a lower conformance layer. First, it is expected that a simpler language is easier to learn. Second, it is expected that a reduced set of language features is easier to implement.

Several proposals for the set of primitives in the language were made. However the motivation given for a particular set of primitives have been rather vague. Since OWL is a novel language we cannot draw on broad practice and motivate a particular set of primitives by common usage.

It is also difficult to evaluate how much effort has to be invested to learn a particular set of primitives. Usually learning is a highly individual process and depends on many other factors such as the previous knowledge of an individual. However, we may assume that the effort is proportional to number of language primitives. This argument would speak for a very small number of primitives. However, the language should be sufficiently more expressive than RDF Schema, whose lack of expressivity gave reason to develop OWL initially.

We therefore argue that the only sensible argument for the set of primitives is the effort required to implement the language. Additionally many vendors

may intend to reuse existing tools for data (and knowledge) representation to lower the cost of an implementation.

This document therefore investigates how a subset of OWL could be implemented on top of available rule-based systems in particular recursion-enabled relational databases. We take an incremental approach and start with a language whose primitives correspond to the primitives available in RDF Schema for domain modelling. This language OWL_0^{Lite} is then incrementally extended with further features. We will have a look at the resulting interaction of the particular set of primitives.

This proposal is structured as follows. First, we introduce the basic formalism and notation employed for describing rule bases. Then we state the fundamental assumptions which direct our construction of OWL Lite. In the next section, the basic representation of RDF data in rule-based systems is presented. Building on that we start with an incremental construction of OWL^{Lite}. We conclude by recapitulating the feature set that can be safely implemented on recursion-enabled databases and in logic programming systems.

2 Basics

The basics required for the further understanding of this document are briefly recapitulated in this section for those readers not fluent in database theory.

2.1 SQL and relational algebra

SQL has emerged as the preeminent query language for commercial relational DBMS's. Since there are numerous dialects of SQL we restrict our attention to the subset of the *query* language that can be formally specified. Early works in database theory proposed several relational algebrae and calculi for this purpose. Codd's seminal paper "Relational completeness of database sublanguages" proofed the equivalence of these approaches and coined the term of *relational completeness*. SQL-92 is relationally complete in the sense that it can express all queries expressive in the relational calculus and algebra.

2.2 Datalog

Another, alternative formal specification for queries is *datalog programs* which have close relations to logic programming environments. This offers the benefit that it is immediately clear how the results developed in this document can be applied in logic programming environments such as Prolog. The main difference between logic-programming (LP) environments and datalog is, that LP permits function symbols, and Datalog does not [1]. Datalog additionally requires that variables available in rule heads must appear in rule bodies.

Logic Programs have a direct translation to predicate logic. All variables are universally qualified by default.

[1] with the exception of constants, which are functions of arity 0

2.3 Datalog in a nutshell

Our presentation follows the lecture slides of [Cho01].[2] The basic elements of the language are predicates and terms. Terms can be either constant symbols such as names (so-called *atomic values*) and numbers (integer, float ...) or logical variable symbols. Usually variables and constants are distinguished by syntactic convention. In the following all variables will start with an upper-case letter. Each logical variable is a placeholder for another value, which can be instantied by substituting it by another term. However, it cannot be assigned directly.

Datalog programs are composed of goals, queries and implications (also called *clauses*). Goals are syntactically represented by a predicate:

$$P(T_1, ..., T_n)$$

This states that predicate P is true of terms T_1 and all other T_i. Queries are syntactically represented as a set of goals:

$$E_1, E_2, ..., E_n$$

This retrieves all E_i. Clauses are syntactically written like implications in Prolog:

$$E_0 : -E_1, ..., E_n.$$

This means, that E_0 must be true, if all E_i are also true. E_0 is also called the head of the clause, while the remaining E_i are called the body of the clause. The body of the clause may be empty. Then the clause is called a *fact*. The clause is also referred to as a *rule* if the body is nonempty. We speak of *recursive rules* if a predicate appears in both the head and the body of a rule.

2.4 Variants of Datalog

In research several variants of Datalog have been proposed. For example, a variant without recursion and extended with negation is $nr - datalog^{\neg}$, which is equivalent to calculus and algebra. $nr - datalog^{\neg}$ programs can easily be simulated using SQL-92 (see [Ull] for a formal proof). Intuitively, this result follows from the following substitutions:

- Each Datalog-rule can be simulated using the select-from-where construct of SQL.
- Multiple rules defining the same predicate can be simulated using *union*.
- Negation in rule bodies can be simulated using *not in*

[2] A more detailed description of Datalog is also available in an online book [Vor97], however it uses a different terminology.

The latest standard of SQL, SQL:1999, also allows recursive queries. Here, use of negation is limited, since the use negation is restricted to stratified databases[KHA89]. Informally this means that, if the definition of a predicate P depends on knowing the complement of a predicate Q, then the definition of Q must not depend on P. In particular, P cannot depend on its own complement. Hence, there is no recursion through negation, so the number of applications of negation is bounded. SQL:1999 can simulate stratifiable Datalog programs.

3 Assumptions

3.1 Let's stay monotonic

The semantics of negation in stratified Datalog is not compatible with the semantics of negation in Description Logics. In stratified Datalog negation builds on the close world assumption. This is centered on a boolean logic, where only the states *true* and *false* exist.

OWL relies on the open world assumption, here the additional state of unknown is available. Therefore negation cannot be computed using the complement (so-called *negation as failure*) as it is done in stratified Datalog. Alternative, more compatible semantics for negation in Datalog exist, namely well-founded semantics, that also relies on such a three-valued logics. However, this is not semantics available in SQL:1999 compliant databases. Well-founded semantics only regards the minimal model when computing negation, whereas Description Logics regard all possible models. Hence, they are also incompatible.

Therefore, SQL:1999 capable databases cannot reason over any OWL primitives which involve the computation of complements. As a consequence we rely on Datalog without negation for the construction of OWL^{Lite}. Hence, only *monotonic reasoning* is used for the OWL^{Lite} conformance layer.

3.2 Uniqueness of Names

Datalog takes the unique names assumption. This is not done in OWL. Without further steps the entailments provided by OWL would therefore be different. We take further steps and rely on the fact that we can only get what is entailed via queries. The steps involve the generation of implicit facts via rules and offering three binary predicates equivalentTo, subClassOf and subPropertyOf to entail equivalence and the subsumptions.

3.3 Implementation Process

We employ a very simple assumption, namely that we implement the semantics of OWL^{Lite} by compilation. Hence, the syntax is parsed and an appropriate (rule-based) knowledge base is compiled. In this compilation process several rule patterns are instantiated. A set of rule pattern captures the semantics of the individual OWL statetements. For example, each subClassOf statement is

translated to a rule that captures the semantics of the particular subClassOf statement.

4 Representation of RDF in rule-based systems

OWL ontologies are syntactically represented in RDF. This would allow a very simple form of representation, which maps each RDF statement to one logical ternary predicate $statement(subject, predicate, object)$. Subject, predicate are RDF resources and the object is either a resource or a literal. This a vertical form of representation which is often chosen for representing sparse data with a large number of attributes such as found in many e-Commerce or digital library scenarios [ASX01].

The usage of one single ternary relation for storage of directed labeled-graphs such as RDF seems to be the most simple solution. On the other hand each resource-value pair could be stored in a separate binary relation on a per property basis. The evaluation of [ASX01] suggests that this representation also slightly outperforms the naïve ternary representation.

Representing data

RDF statements are written as binary predicates. Here the predicate of the statement is the name of the Datalog predicate. Hence, if (a, b) instance of property P we write the fact

$$P(a, b).$$

In our approach this representation is extended with additional unary relations which are used to store the class-individual relationship, where a separate predicate is created for each class. Hence, additionally to writing $type(a, c)$. facts to say that a is an instance of c, we write the fact

$$C(a).$$

We assume for unnamed RDF individuals, that the required symbols are created on the fly. A system could use several means for (re)identifying such anonymous individuals, e.g. a naming scheme.

Representing the ontology

We cannot ask queries about the ontology level in Datalog, since this would involve talking about predicates, when using the above-mentioned representation. Hence, we employ a *dual* representation, i.e. additional constants are created for each property and class as well as facts that say that the constants are of their respective kind. This is similar to the catalogue components of relational databases, which store information about the available relations and their attributes.

Hence, we have a fact

$$Class(c).$$

which states, that c is a class and a fact

$$Property(p).$$

which states, that p is a property. Please note, that artificial identifiers have to be created for all unnamed classes, i.e. when using property restrictions. We will have to talk about individuals later, hence we will need a metaclass individual in our internal representation of OWL. This metaclass is populated via the following rule:

$$Individual(X) : -type(X, C), Class(C).$$

The population of the Individual predicate is purely intentional, since there is no such syntactic primitive in OWL.

5 Construction of OWL^{Lite}

5.1 OWL_0^{Lite} - RDF Schema

Here, we focus only on the domain modelling capabilites of RDF Schema. Hence we do not care about the metamodelling capabilites of RDF Schema. Therefore, if someone subclasses the RDF Schema property subclassof, the semantics of this statement is not known to the system, e.g. it will not know that this is also a transitive property.

The domain modelling part of RDF Schema provides have four primitives, namely

- subclassof: For each (C rdfs:subclassof D) statement the following rule pattern is instantiated
$$D(X) : -C(X).$$
- subpropertyof: For each (M rdfs:subpropertyof N) statement the following rule pattern is instantiated
$$N(X, Y) : -M(X, Y).$$
- domain: (P rdfs:domain C) is translated to rules of the following form
$$C(X) : -P(X, Y).$$
- range: (P rdfs:range C) is translated to rules of the following form
$$C(Y) : -P(X, Y).$$

Additionally the transitivity of the subClassOf and subPropertyOf predicates must be represented once via the following pair of rules.

$$subClassOf(X, Z) : -subClassOf(X, Y), subClassOf(Y, Z).$$

$$subPropertyOf(X, Z) : -subPropertyOf(X, Y), subPropertyOf(Y, Z).$$

5.2 OWL_1^{Lite} - Toulouse proposal

The so-called Toulouse proposal was a joint proposal by several members of the WebOnt working group. The name is drawn on the meeting location of those people, namely the 2002 conference on Knowledge Representation, which was located in the French town of Toulouse. In addition to the RDF Schema primitives described above it provides means for stating :

- the (in)equality of individuals
- additional property characteristics
- functionality of properties

5.2.1 Equality

Equality for classes and properties is already provided via cyclic subclassof and subpropertyof definitions. Additionally OWL_1^{Lite} provides additional syntactic shortcuts for stating this kind of equivalence. The semantics of those primitives are translated as using the following rule patterns:

- sameClassAs: For each (C owl:sameClassAs D) the following rule pattern is instantiated:

$$D(X) : -C(X).$$

$$C(X) : -D(X).$$

- samePropertyAs: For each (M owl:samePropertyAs N) the following rule pattern is instantiated:

$$N(X,Y) : -M(X,Y).$$

$$M(X,Y) : -N(X,Y).$$

We can infer the equivalence of classes and properties easily using the following four rules:

$$sameClassAs(X,Y) : -subClassOf(X,Y), subClassOf(Y,X).$$

$$sameClass(X,X) : -Class(X).$$

$$samePropertyAs(X,Y) : -subPropertyOf(X,Y), subPropertyOf(Y,X).$$

$$samePropertyAs(X,X) : -Property(X).$$

Equivalence of individuals is provided via the *sameIndividualAs* primitive. The following set of rules captures the semantics of the predicate:

- Algebraic properties: since it is an equivalence relation
  - symmetry:

$$sameIndividualAs(X, Y) : -sameIndividualAs(Y, X).$$

  - transitivity:

$$sameIndividualAs(X, Z) : -sameIndividualAs(X, Y),$$
$$sameIndividualAs(Y, Z).$$

  - reflexivity:

$$sameIndividualAs(X, X) : -Individual(X).$$

- Class and Property Membership: Equivalent individuals are also part of the respective extensions.
  - Class membership:

$$C(X) : -C(Y), sameIndividualAs(X, Y).$$

  - Property membership:

$$P(X, Z) : -P(X, Y), sameIndividualAs(Y, Z).$$
$$P(Y, Z) : -P(X, Z), sameIndividualAs(X, Y).$$

EquivalentTo is an additional syntactic shortcut is provided to state that something is equivalent, namely equivalentTo. Its semantics can be captured as follows:

$$equivalentTo(X, Y) : -sameClassAs(X, Y).$$
$$equivalentTo(X, Y) : -samePropertyAs(X, Y).$$
$$equivalentTo(X, Y) : -sameIndividualAs(X, Y).$$
$$sameClassAs(X, Y) : -equivalentTo(X, Y), Class(X), Class(Y).$$
$$samePropertyAs(X, Y) : -equivalentTo(X, Y), Property(X), Property(Y).$$
$$sameIndividualAs(X, Y) : -equivalentTo(X, Y), Individual(X), Individual(Y).$$

The use of equivalentTo is restricted in the language, it is not possible to cross the partition in the language between classes, properties and individuals.

$$inconsistent(X, Y) : -Class(X), Property(Y), equivalentTo(X, Y).$$
$$inconsistent(X, Y) : -Individual(X), Property(Y), equivalentTo(X, Y).$$
$$inconsistent(X, Y) : -Individual(X), Class(Y), equivalentTo(X, Y).$$

Inequality of individuals The differentIndividualFrom primitive can be used to denote that two individuals are not the same. This is required, since OWL_1^{Lite} may infer that some individuals may be the same. Hence, an inconsistency arises, if we find out that both facts are stated. This is captured via the following rule:

$$inconsistent(X, Y) : -sameIndividualAs(X, Y), differentIndividualFrom(X, Y)$$

!!! To be discussed !!! Please note that we can only ask whether two individuals are equivalent or not. The answer 'unknown', which must be returned, whenever we neither have a fact, that says that the individuals are equivalent nor have a fact, that says that the individuals are different, would have to be supported using some further rule. However this rule would require negation.

5.2.2 Property Characteristics

OWL_1^{Lite} Allows to state further property characteristics. Syntactically the properties that are subject of those additional characteristics are represented as subclasses of ObjectProperties. ObjectProperties are in turn subclasses of rdf:Property. These statements are part of the metalanguage of OWL_1^{Lite} and treated represented like rdf:Property via binary predicates in the database and unary predicates to capture the Property membership.

However, further rule patterns are instantiated for each property P:

- TransitiveProperty:

$$P(X, Z) : -P(X, Y), P(Y, Z)$$

- SymmetricProperty:

$$P(X, Y) : -P(Y, X)$$

5.2.3 InverseProperties

OWL_1^{Lite} allows to specify that the values of two ObjectProperties are inverse to each other. Hence the semantics of all (P owl:inverseOf R) statements has to be captured via the instantiation of the following rule patterns :

$$P(X, Y) : -R(Y, X)$$

$$R(X, Y) : -P(Y, X)$$

5.2.4 FunctionalProperty

Functional properties are properties that are stated to have a unique value. If a property is a FunctionalProperty, then it has no more than one value. It may have no values. Another way of saying this is that the property's minimum cardinality is zero and its maximum cardinality is 1.

This can be represented in Datalog as follows:

$$sameIndividualAs(X, Y) : -P(A, X), P(A, Y)$$

Hence, it is entailed that X and Y must be equivalent instances. If there is a statement, which declares them to be different from each other, the knowledge base is in an inconsistent state.

5.3 OWL_2^{Lite} - Current OWL Lite proposal

5.3.1 InverseFunctional

A property of type InverseFunctioanlProperty[3] is a subclass of ObjectProperty. If a property is of this type, then the inverse of the property is functional. Thus the inverse of the property has at most one value.

This can be represented in Datalog by the following rule pattern:

$$sameIndividualAs(X, Y) : -P(X, A), P(Y, A)$$

Hence, it is entailed that X and Y are equal. If there is a statement, which declares them to be different from each other, the knowledge base is inconsistent.

5.3.2 Cardinality Constraints

OWL_2^{Lite} provides further means to restrict the cardinality of properties. The values are restricted to the values 0 and 1. The owl:minCardinality construct is used to denote the minimum cardinality of a property. The owl:maxCardinality construct is used to denote the maximum cardinality of a property. The owl:cardinality construct is a convenience constructor for setting both owl:minCardinalty and owl:maxCardinality to the same value.

Cardinality 0,1 : Saying that a property has minimumCardinalty of 0 and maximum Cardinality of 1 is saying that a property is functional. It is therefore translated to the instatiation of the rule pattern stated there.

Cardinality 0,0 : This means that a property may not be instantiated. Hence an inconsistency follows from having any property value on P:

$$inconsistent(X, Y) : -P(X, Y).$$

Minimum Cardinality 1 : This is not possible to say due to the explicit universal quantification of all variables.

In predicate logic the minimum cardinality of one would be specified by use of the existential quantifier:

$$\forall X \exists Y : P(X, Y)$$

Hence, this information cannot be captured by Datalog.

[3]This type of property was previously called unambiguous property and IsTheOnlyOne property.

5.3.3 Local range restrictions

A property on a particular class may have a local range restriction associated with it.

allValuesFrom : This means that if an individual instance of the class is related by the property to a second individual, then the second individual can be inferred to be an instance of the local range restriction class.

May C be the local range restriction on property P for class D. This can be captured via rules of the following form

$$C(Y) : -P(X, Y), D(X).$$

someValuesFrom : This means that a particular class may have a restriction on a property that at least one value for that property is of a certain type. This is existential by nature and can therefore not be captured by Datalog. It can be captured in first-order logic as follows.

$$\forall X \exists Y : C(Y) \leftarrow P(X, Y) \wedge D(X)$$

5.4 OWL_n^{Lite} - Full Owl

5.4.1 hasValue

Has Value allows to define a class via a certain property value. It can be supported easily. Let v the value of property P, which constitutes the class C, then the following rule pattern can capture this semantics:

$$C(X) : -P(X, v).$$
$$P(X, v) : -C(X).$$

5.4.2 Full cardinality

Cannot be supported, since even restricted cardinality cannot be supported. Non-binary values cannot be captured at all, since we are not able to count in general Datalog.

5.4.3 Set construction of classes

This is problematic with respect to equivalence and the absence of disjunction in the head for the case of disjunction. However, conjunction ($D \equiv C_1 \sqcap C_2 \sqcap ... \sqcap C_n$) could be supported easily via the following pattern:

$$D(X) : -C_1(X), C_2(X), ..., C_n(X).$$
$$C_1(X) : -D(X).$$
$$C_2(X) : -D(X).$$
$$C_n(X) : -D(X).$$

5.4.4 Construction of classes by enumeration

This is problematic, since we cannot disallow using predicates in rules.

6 Conclusion

Following the previous discussion the OWL Lite proposal should be changed to OWL_1^{Lite} and may be augmented safely with the InverseFunctional property as well as universal local range restrictions (allValuesFrom) from OWL_2^{Lite}. Additionally we can easily support conjunction of classes and the hasValue property restriction.

Datalog can support the following features in an OWL^{Lite} conformance layer:

1. Primitive classes
2. rdfs:subClassOf
3. rdfs:subpropertyof
4. rdfs:domain
5. rdfs:range
6. rdf:Property
7. (un)named individuals
8. owl:sameClassAs
9. owl:samePropertyAs
10. owl:sameIndividualAs
11. owl:equivalentTo
12. owl:differentIndividualAs
13. owl:ObjectProperty
14. owl:inverseOf
15. owl:TransitiveProperty
16. owl:SymmetricProperty
17. owl:FunctionalProperty
18. owl:InverseFunctionalProperty
19. owl:allValuesFrom
20. owl:hasValue
21. owl:intersectionOf

Acknowledgements This work was supported by the E.U. through the WonderWeb IST project. I would like to thank Benjamin Grossof, Ian Horrocks, Frank van Harmelen, Daniel Oberle, Steffen Staab and Rudi Studer for insightful feedback and comments.

References

[Ari] Aristoteles. *Nicomachean Ethics, Book VI.* The Internet Classics Archive, http://classics.mit.edu/Aristotle/nicomachaen.6.vi.html.

[ASX01] Rakesh Agrawal, Amit Somani, and Yirong Xu. Storage and querying of e-commerce data. In *The VLDB Journal*, pages 149–158, 2001.

[Cho01] Jan Chomicki. Datalog slides. CSE 720 Seminar: Information Integration, University of Buffalo, Departement of Computer Science and Engineering, Internet: http://www.cs.buffalo.edu/~chomicki/cse7xx.html, September 2001.

[KHA89] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a theory of declarative knowledge. *Foundations of Deductive Databases and Logic Programming*, pages 89–148, 1989.

[Ull] Jeffrey D. Ullman. *Principles of Database and Knowledge-base Systems*, volume 1. Computer Science Press.

[Vor97] Voronkov. Logic databases. Course, University of Uppsala, Departement of Computer Science, Internet: http://user.it.uu.se/~voronkov/ddb.htm, 1997.