DR-Prolog: A System for Reasoning with Rules and Ontologies on the Semantic Web

Grigoris Antoniou and Antonis Bikakis

Institute of Computer Science, FO.R.T.H
Vassilika Vouton, P.O. Box 1385, GR 71110, Heraklion, Greece
{antoniou,bikakis}@ics.forth.gr

Abstract

Defeasible reasoning is a rule-based approach for efficient reasoning with incomplete and inconsistent information. Such reasoning is, among others, useful for ontology integration, where conflicting information arises naturally; and for the modeling of business rules and policies, where rules with exceptions are often used. This paper describes these scenarios in more detail, and reports on the implementation of a system for defeasible reasoning on the Web. The system (a) is syntactically compatible with RuleML; (b) features strict and defeasible rules, priorities and two kinds of negation; (c) is based on a translation to logic programming with declarative semantics; (d) is flexible and adaptable to different intuitions within defeasible reasoning; and (e) can reason with rules, RDF, RDF Schema and (parts of) OWL ontologies.

Introduction

The development of the Semantic Web (Berners Lee *et al.,* 2001) proceeds in layers, each layer being on top of other layers. At present, the highest layer that has reached sufficient maturity is the ontology layer in the form of the description logic based languages of DAML+OIL (Connolly *et al.*, 2001) and OWL (Dean and Schreiber, 2004).

The next step in the development of the Semantic Web will be the logic and proof layers that will offer enhanced representation and reasoning capabilities. *Rule systems* appear to lie in the mainstream of such activities. Moreover, rule systems can also be utilized in ontology languages. So, in general rule systems can play a twofold role in the Semantic Web initiative: (a) they can serve as extensions of, or alternatives to, description logic based ontology languages; and (b) they can be used to develop declarative systems on top of (using) ontologies. Reasons why rule systems are expected to play a key role in the further development of the Semantic Web include the following:

- Seen as subsets of predicate logic, monotonic rule systems (Horn logic) and description logics are orthogonal; thus they provide additional expressive power to ontology languages.
- Efficient reasoning support exists to support rule languages.
- Rules are well known in practice, and are reasonably well integrated in mainstream information technology.

Possible interactions between description logics and monotonic rule systems were studied in (Grosof *et al.*, 2003). Based on that work and on previous work on hybrid reasoning (Levy and Rousset, 1998) it appears that the best one can do at present is to take the intersection of the expressive power of Horn logic and description logics; one way to view this intersection is the Horn-definable subset of OWL.

This paper is devoted to a different problem, namely *conflicts among rules*. Here we just mention the main sources of such conflicts, which are further expanded in the next section. At the ontology layer:

- Default inheritance within ontologies
- Ontology merging

And at the logic and reasoning layers:

- Rules with exceptions as a natural representation of business rules
- Reasoning with incomplete information

Defeasible reasoning is a simple rule-based approach to reasoning with incomplete and inconsistent information. It can represent facts, rules, and priorities among rules. This reasoning family comprises defeasible logics (Nute, 1994; Antoniou *et al.*, 2001) and Courteous Logic Programs (Grosof 1997). The main advantage of this approach is the combination of two desirable features: enhanced representational capabilities allowing one to reason with incomplete and contradictory information, coupled with low computational complexity compared to mainstream nonmonotonic reasoning.

In this paper we report on the implementation of a defeasible reasoning system for reasoning on the Web. Its main characteristics are the following:

- Its user interface is compatible with RuleML (RuleML), the main standardization effort for rules on the Semantic Web.
- It is based on Prolog. The core of the system consists of a well-studied translation (Antoniou *et. al*., 2001) of defeasible knowledge into logic programs under Well-Founded Semantics (van Gelder *et al.*, 1991). This declarative translation distinguishes our work from other implementations (Grosof *et al.*, 2002; Maher *et al.*, 2001).
- The main focus is on flexibility. Strict and defeasible rules and priorities are part of the interface and the implementation. Also, a number of variants were implemented (ambiguity blocking, ambiguity

propagating, conflicting literals; see below for further details).

- The system can reason with rules and ontological knowledge written in RDF Schema (RDFS) or OWL.

As a result of the above, DR-Prolog is a powerful declarative system supporting:

- rules, facts and ontologies
- all major Semantic Web standards: RDF, RDFS, OWL, RuleML
- monotonic and nonmonotonic rules, open and closed world assumption, reasoning with inconsistencies.

The paper is organized as follows. The next section describes the main motivations for conflicting rules on the Semantic Web. The third section describes the basic ideas of defeasible reasoning, and the forth one describes the translation of defeasible logic, and of RDF, RDFS and (parts of) OWL into logic programs. The fifth section reports on the implemented system. The sixth section discusses related work, and the last section concludes with a summary and some ideas for future work.

Motivation for Nonmonotonic Rules on the Semantic Web

We believe that we have to distinguish between two types of knowledge on the Semantic Web. One is static knowledge, such as factual and ontological knowledge which contains general truths that do not change often. And the other is dynamic knowledge, such as business rules, security policies etc. that change often according to business and strategic needs. The first type of knowledge requires monotonic reasoning based on an open world assumption to guarantee correct propagation of truths. But for dynamic knowledge flexible, context-dependent and inconsistency tolerant nonmonotonic reasoning is more appropriate for drawing practical conclusions.
Obviously, a combination of both types of knowledge is required for practical systems. Defeasible logic, as described in the next section, supports both kinds of knowledge. Before presenting its technical details, we motivate the use of nonmonotonic rules in more detail.

Reasoning with Incomplete Information*:* Antoniou and Arief (2002) describe a scenario where business rules have to deal with incomplete information: in the absence of certain information some assumptions have to be made which lead to conclusions that are not supported by classical predicate logic. In many applications on the Web such assumptions must be made because other players may not be able (e.g. due to communication problems) or willing (e.g. because of privacy or security concerns) to provide information. This is the classical case for the use of nonmonotonic knowledge representation and reasoning (Marek and Truszczynski, 1993).

Rules with Exceptions*:* Rules with exceptions are a natural representation for policies and business rules (Antoniou *et. al*, 1999). And priority information is often implicitly or explicitly available to resolve conflicts among rules. Potential applications include security policies (Ashri *et al.*, 2004; Li *et al.*, 2003), business rules (Antoniou and Arief 2002), personalization, brokering, bargaining, and automated agent negotiations (Governatori *et al.*, 2001).

Default Inheritance in Ontologies*:* Default inheritance is a well-known feature of certain knowledge representation formalisms. Thus it may play a role in ontology languages, which currently do not support this feature. Grosof and Poon (2003) present some ideas for possible uses of default inheritance in ontologies. A natural way of representing default inheritance is rules with exceptions, plus priority information. Thus, nonmonotonic rule systems can be utilized in ontology languages.

Ontology Merging*:* When ontologies from different authors and/or sources are merged, contradictions arise naturally. Predicate logic based formalisms, including all current Semantic Web languages, cannot cope with inconsistencies.

If rule-based ontology languages are used and if rules are interpreted as defeasible (that is, they may be prevented from being applied even if they can fire) then we arrive at nonmonotonic rule systems. A skeptical approach, as adopted by defeasible reasoning, is sensible because it does not allow for contradictory conclusions to be drawn. Moreover, priorities may be used to resolve some conflicts among rules, based on knowledge about the reliability of sources or on user input. Thus, nonmonotonic rule systems can support ontology integration.

Defeasible Logics

Basic Characteristics

The root of defeasible logics lies on research in knowledge representation, and in particular on inheritance networks. Defeasible logics can be seen as inheritance networks expressed in a logical rules language. In fact, they are the first nonmonotonic reasoning approach designed from its beginning to be implementable.
Being nonmonotonic, defeasible logics deal with potential conflicts (inconsistencies) among knowledge items. Thus they contain classical negation, contrary to usual logic programming systems. They can also deal with negation as failure (NAF), the other type of negation typical of nonmonotonic logic programming systems; in fact, Wagner (2003) argues that the Semantic Web requires both types of negation. In defeasible logics, often it is assumed that NAF is not included in the object language. However, as Antoniou *et al.* (2000a) show, it can be easily simulated when necessary. Thus, we may use NAF in the object language and transform the original knowledge to logical rules without NAF exhibiting the same behavior.
Conflicts among rules are indicated by a conflict between their conclusions. These conflicts are of local nature. The simpler case is that one conclusion is the negation of the other. The more complex case arises when the conclusions have been declared to be mutually exclusive,

a very useful representation feature in practical applications.

Defeasible logics are skeptical in the sense that conflicting rules do not fire. Thus consistency of drawn conclusions is preserved.

Priorities on rules may be used to resolve some conflicts among rules. Priority information is often found in practice, and constitutes another representational feature of defeasible logics.

The logics take a pragmatic view and have low computational complexity. This is, among others, achieved through the absence of disjunction and the local nature of priorities: only priorities between conflicting rules are used, as opposed to systems of formal argumentation where often more complex kinds of priorities (e.g. comparing the strength of reasoning chains) are incorporated.

Generally speaking, defeasible logics are closely related to Courteous Logic Programs (Grosof, 1997); the latter were developed much later than defeasible logics. DLs have the following advantages:

- They have more general semantic capabilities, e.g. in terms of loops, ambiguity propagation etc.
- They have been studied much more deeply, with strong results in terms of proof theory (Antoniou *et al.*, 2001), semantics (Maher, 2002) and computational complexity (Maher, 2001). As a consequence, its translation into logic programs, a cornerstone of DR-Prolog, has also been studied thoroughly (Maher *et al.*, 2001; Antoniou and Maher, 2002).

However, Courteous Logic Programs have also had some advantages:

- They were the first to adopt the idea of mutually exclusive literals, an idea incorporated in DR-Prolog.
- They allow access to procedural attachments, something we have chosen not to follow in our work so far.

Syntax

A *defeasible theory D* is a triple (*F,R,*>) where *F* is a finite set of facts, *R* a finite set of rules, and > a superiority relation on R. In expressing the proof theory we consider only propositional rules. Rules containing free variables are interpreted as the set of their variable-free instances.

There are two kinds of rules (fuller versions of defeasible logics include also defeaters): *Strict rules* are denoted by

$A \rightarrow p$,

and are interpreted in the classical sense: whenever the premises are indisputable then so is the conclusion. An example of a strict rule is "Professors are faculty members". Written formally:

$professor(X) \rightarrow faculty(X)$.

Inference from strict rules only is called *definite inference*. Strict rules are intended to define relationships that are definitional in nature, for example ontological knowledge.

Defeasible rules are denoted by

$A \Rightarrow p$,

and can be defeated by contrary evidence. An example of such a rule is

$faculty(X) \Rightarrow tenured(X)$

which reads as follows: "Professors are typically tenured".

A *superiority relation* on R is an acyclic relation > on R (that is, the transitive closure of > is irreflexive). When

$r_1 > r_2$,

then r_1 is called *superior* to r_2, and r_2 *inferior* to r_1. This expresses that r_1 may override r_2. For example, given the defeasible rules

$r: professor(X) \Rightarrow tenured(X)$

$r': visiting(X) \Rightarrow \neg tenured(X)$

which contradict one another: no conclusive decision can be made about whether a visiting professor is tenured. But if we introduce a superiority relation > with

$r' > r$,

then we can indeed conclude that a visiting professor cannot be tenured.

A formal definition of the proof theory is found in (Antoniou *et al.*, 2001).

Simulation of Negation As Failure in the Object Language

We follow a technique based on auxiliary predicates first presented in (Antoniou *et al.*, 2000a), but which is often used in logic programming. According to this technique, a defeasible theory with NAF can be modularly transformed into an equivalent one without NAF. Every rule

$r: L_1,\ldots,L_n, \sim M_1,\ldots, \sim M_k \Rightarrow L$

where $L_1,\ldots,L_n, M_1,\ldots,M_k$ are atoms and $\sim M_i$ denotes the weak negation of M_i, is replaced by the rules:

$r: L_1,\ldots,L_n, neg(M_1),\ldots,neg(M_k) \Rightarrow L$

$\Rightarrow neg(M_1)$

…

$\Rightarrow neg(M_k)$

$M_1 \Rightarrow \neg neg(M_1)$

…

$M_k \Rightarrow \neg neg(M_k)$

where $neg(M1),\ldots,neg(Mk)$ are new auxiliary atoms and $\neg neg(M_i)$ denotes the strong negation of M_i. If we restrict attention to the original language, the set of conclusions remains the same.

Ambiguity Blocking and Ambiguity Propagating Behavior

A literal is *ambiguous* if there is a chain of reasoning that supports a conclusion that *p* is true, another that supports that ¬*p* (where ¬*p* denotes strong negation of *p*) is true, and the superiority relation does not resolve this conflict. We can illustrate the concept of ambiguity propagation through the following example.

$r_1: quaker(X) \Rightarrow pacifist(X)$

$r_2: republican(X) \Rightarrow \neg pacifist(X)$

$r_3: pacifist(X) \Rightarrow \neg hasGun(X)$

$r_4: livesInChicago(X) \Rightarrow hasGun(X)$

$quaker(a)$

$republican(a)$

$livesInChicago(a)$

$r_3 > r_4$

Here `pacifist(a)` is ambiguous. The question is whether this ambiguity should be propagated to the dependent literal `hasGun(a)`. In one defeasible logic variant it is detected that rule r_3 cannot fire, so rule r_4 is unopposed and gives the defeasible conclusion `hasGun(a)`. This behavior is called *ambiguity blocking*, since the ambiguity of `pacifist(a)` has been used to block r_3 and resulted in the unambiguous conclusion `hasGun(a)`.

On the other hand, in the ambiguity propagation variant, although rule r_3 cannot lead to the conclusion `hasGun(a)` (as `pacifist(a)` is not provable), it opposes rule r_4 and the conclusion `hasGun(a)` cannot also be drawn.

This question has been extensively studied in artificial intelligence, and in particular in the theory of inheritance networks. A preference for ambiguity blocking or ambiguity propagating behavior is one of the properties of nonmonotonic inheritance nets over which intuitions can clash. Ambiguity propagation results in fewer conclusions being drawn, which might make it preferable when the cost of an incorrect conclusion is high. For these reasons an ambiguity propagating variant of DL is of interest.

Conflicting Literals

Usually in Defeasible Logics only conflicts among rules with complementary heads are detected and used; all rules with head L are considered as *supportive* of L, and all rules with head $\neg L$ as *conflicting*. However, in applications often literals are considered to be conflicting, and at most one of a certain set should be derived. For example, the risk an investor is willing to accept may be classified in one of the categories low, medium, and high. The way to solve this problem is to use a constraint rule of the form

```
conflict :: low, medium, high
```

Now if we try to derive the conclusion `high`, the conflicting rules are not just those with head ¬`high`, but also those with head `low` and `medium`. Similarly, if we are trying to prove ¬`high`, the supportive rules include those with head `low` or `medium`.

In general, given a `conflict::L,M`, we augment the defeasible theory by:

r_i: $q_1, q_2, \ldots, q_n \rightarrow \neg L$
  for all rules r_i: $q_1, q_2, \ldots, q_n \rightarrow M$
r_i: $q_1, q_2, \ldots, q_n \rightarrow \neg M$
  for all rules r_i: $q_1, q_2, \ldots, q_n \rightarrow L$
r_i: $q_1, q_2, \ldots, q_n \Rightarrow \neg L$
  for all rules r_i: $q_1, q_2, \ldots, q_n \Rightarrow M$
r_i: $q_1, q_2, \ldots, q_n \Rightarrow \neg M$
  for all rules r_i: $q_1, q_2, \ldots, q_n \Rightarrow L$

The superiority relation among the rules of the defeasible theory is propagated to the "new" rules.

Translation into Logic Programs

Translation of Defeasible Theories

The translation of a defeasible theory D into a logic program $P(D)$ has a certain goal: to show that

p is defeasibly provable in $D \Leftrightarrow$
p is included in the Well-Founded Model of $P(D)$

Two different translations have so far been proposed, sharing the same basic structure:

The translation of (Antoniou *et al.*, 2000b; Maher *et al.*, 2001) where a meta-program was used.

The translation of (Antoniou and Maher, 2002), which makes use of control literals.

It is an open question which is better in terms of computational efficiency, although we conjecture that for large theories the meta-program approach is better, since in the other approach a large number of concrete program clauses is generated. Therefore, we have adopted this approach in our implementation.

Translation of Ambiguity Blocking Behavior. The metaprogram which corresponds to the ambiguity blocking behavior of the defeasible theories consists of the following program clauses:

The first three clauses define the class of rules used in a defeasible theory.

```
supportive_rule(Name,Head,Body):-
  strict(Name,Head,Body).
supportive_rule(Name,Head,Body):-
  defeasible(Name,Head,Body).
rule(Name,Head,Body):-

  supportive_rule (Name,Head,Body).
```

The following clauses define the definite provability: a literal is definitely provable if it is a fact or is supported by a strict rule, the premises of which are definitely provable.

```
definitely(X):- fact(X).
definitely(X):-strict(R,X,L),
  definitely_provable(L).
definitely_provable([]).
definitely_provable(X):- definitely(X).
definitely_provable([X1|X2]):-
  definitely_provable(X1),
  definitely_provable(X2).
```

The next clauses define the defeasible provability: a literal is defeasibly provable, either if it is definitely provable, or if its complementary is not definitely provable, and it is supported by a defeasible rule, the premises of which are defeasibly provable, and which is not overruled. The sk_not operator, which we use as the negation operator in the following clauses, is provided by XSB (the logic programming system that stands in the core of DR-Prolog), and allows for correct execution of programs according to the well-founded semantics.

```
defeasibly(X):- definitely(X).
defeasibly(X):- negation(X,X1),
```

```
  supportive_rule(R,X,L),
  defeasibly_provable(L),
  sk_not(definitely(X1)),
  sk_not(overruled(R,X)).
defeasibly_provable([]).
defeasibly_provable(X):- defeasibly(X).
defeasibly_provable([X1|X2]):-
  defeasibly_provable(X1),
  defeasibly_provable(X2).
```

The next clause defines that a rule is overruled when there is a conflicting rule, the premises of which are defeasible provable, and which is not defeated.

```
overruled(R,X):- negation(X,X1),
  supportive_rule(S,X1,U),
defeasibly_provable(U),
  sk_not(defeated(S,X1)).
```

The next clause defines that a rule is defeated when there is a superior conflict rule, the premises of which are defeasibly provable. The last two clauses are used to define the negation of a literal.

```
defeated(S,X):-sup(T,S), negation(X,X1),
  supportive_rule(T,X1,V),
  defeasibly_provable(V).
negation(~(X),X):- !.
negation(X,~(X)).
```

For a defeasible theory $D = (F,R,>)$, where F is the set of the facts, R is the set of the rules, and $>$ is the set of the superiority relations between the rules of the theory, we add facts according to the following guidelines:

```
fact(p).
```
for each $p \in F$
```
strict(r_i,p,[q_1,…,q_n]).
```
for each rule $r: q_1,q_2,\ldots,q_n \rightarrow p \in R$
```
defeasible(r_i,p,[q_1,…,q_n]).
```
for each rule $r: q_1,q_2,\ldots,q_n \Rightarrow p \in R$
```
sup(r,s).
```
for each pair of rules such that $r>s$

Translation of Ambiguity Propagating Behavior. In order to support the ambiguity propagation behavior of a defeasible theory, we only have to modify the program clauses which define when a rule is overruled. In particular, in this variant a rule is overruled when there is a conflicting rule, the premises of which are supported, and which is not defeated.

```
overruled(R,X):- negation(X,X1),
  supportive_rule(S,X1,U),
  supported_list(U),
  sk_not(defeated(S,X1)).
```

The next clauses define that a literal is supported, either if it is definitely provable, or if there is a supportive rule, the premises of which are supported, and which is not defeated.

```
supported(X):- definitely(X).
supported(X):-supportive_rule(R,X,L),
  supported_list(L),
  sk_not(defeated(R,X)).
supported_list([]).
supported_list(X):- supported(X).
supported_list([X1|X2]):-
  supported_list(X1),
  supported_list(X2).
```

Translation of RDF(S) and parts of OWL ontologies

In order to support reasoning with RDF/S and OWL ontologies, we translate RDF data into logical facts, and RDFS and OWL statements into logical facts and rules.
For RDF data, the SWI-Prolog RDF parser (SWI) is used to transform it into an intermediate format, representing triples as

```
rdf(Subject, Predicate, Object).
```

Some additional processing
(i) transforms the facts further into the format

```
Predicate(Subject, Object);
```

(ii) cuts the namespaces and the "comment" elements of the RDF files, except for resources which refer to the RDF or OWL Schema, for which namespace information is retained.
In addition, for processing RDF Schema information, the following rules capturing the semantics of RDF Schema constructs are created:

```
a: C(X):- rdf:type(X,C).
b: C(X):- rdfs:subClassOf(Sc,C),Sc(X).
c: P(X,Y):- rdfs:subPropertyOf(Sp,P),
            Sp(X,Y).
d: D(X):- rdfs:domain(P,D),P(X,Z).
e: R(Z):- rdfs:range(P,R),P(X,Z).
```

Parts of OWL ontologies can also be translated using logical rules, which capture the semantics of some of the OWL constructs.

Equality

```
o1: D(X):- C(X),owl:equivalentClass(C,D).
o2: C(X):- D(X),owl:equivalentClass(C,D).
o3: P(X,Y):- Q(X,Y),
     owl:equivalentProperty(P,Q).
o4: Q(X,Y):- P(X,Y),
     owl:equivalentProperty(P,Q).
o5: owl:equivalentClass(X,Y):-
     rdfs:subClassOf(X,Y),
     rdfs:subClassOf(Y,X).
o6 :owl:equivalentProperty(X,Y):-
     rdfs:subPropertyOf(X,Y),
     rdfs:subPropertyOf(Y,X)
o7 : C(X):- C(Y),
      owl:sameIndividualAs(X,Y).
o8 : P(X,Z):- P(X,Y),
     owl:sameIndividualAs(Y,Z).
o9 : P(Z,Y):- P(X,Y),
      owl:sameIndividualAs(X,Z).
o10: owl:sameIndividualAs(X,Y):-
     owl:sameIndividualAs(Y,X).
o11: owl:sameIndividualAs(X,Z):-
       owl:sameIndividualAs(X,Y),
       owl:sameIndividualAs(Y,Z).
```

```
o12: owl:sameAs(X,Y):-
       owl:equivalentClass(X,Y).
o13: owl:sameAs(X,Y):-
       owl:equivalentProperty(X,Y).
o14: owl:sameAs(X,Y):-
       owl:sameIndividualAs(X,Y).
```

Property Characteristics

```
o15: P(X,Z):- P(X,Y), P(Y,Z),
       rdf:type(P,owl:TransitiveProperty).
o16: P(X,Y):- P(Y,X),
       rdf:type(P,owl:SymmetricProperty).
o17: P(X,Y):- Q(Y,X),owl:Inverseof(P,Q).
o18: Q(X,Y):- P(Y,X),owl:Inverseof(P,Q).
o19: owl:sameIndividualAs(X,Y):-
     P(A,X),P(A,Y),
     rdf:type(P,owl:FunctionalProperty).
o20: owl:sameIndividualAs(X,Y):-
     P(X,A),P(Y,A),
rdf:type(P,owl:InverseFunctionalProperty)
```

Property Restrictions

```
o21: D(Y):- C(X),P(X,Y),
     rdfs:subClassOf(C,R),
     rdf:type(R,owl:Restriction),
     owl:onProperty(R,P),
     owl:allValuesFrom(R,D),
     rdf:type(D,owl:Class).
o22: C(X):- P(X,V),rdfs:subClassOf(C,R),
     rdf:type(R,owl:Restriction),
   owl:onProperty(R,P),owl:hasValue(R,V).
o23: P(X,V):- C(X),rdfs:subClassOf(C,R),
     rdf:type(R,owl:Restriction),
   owl:onProperty(R,P),owl:hasValue(R,V).
```

Collections

```
o24: D(X):- C1(X), C2(X),
     owl:IntersectionOf(D,Collect),
     rdf:type(Collect,Collection),
     memberOf(C1,Collect),
     memberOf(C2,Collect).
o25: C1(X):- D(X),
     owl:IntersectionOf(D,Collect),
     rdf:type(Collect,Collection),
     memberOf(C1,Collect),
     memberOf(C2,Collect).
o26: C2(X):- D(X),
     owl:IntersectionOf(D,Collect),
     rdf:type(Collect,Collection),
     memberOf(C1,Collect),
     memberOf(C2,Collect).
o27: C(X):- owl:oneOf(C,Collect),
     rdf:type(Collect,Collection),
     memberOf(X,Collect).
```

Implementation

DR-Prolog, in accordance with the general philosophy of logic programming, is designed to answer queries. In fact, there are two kinds of queries, depending on which strength of proof we are interested in: definite or defeasible provability.

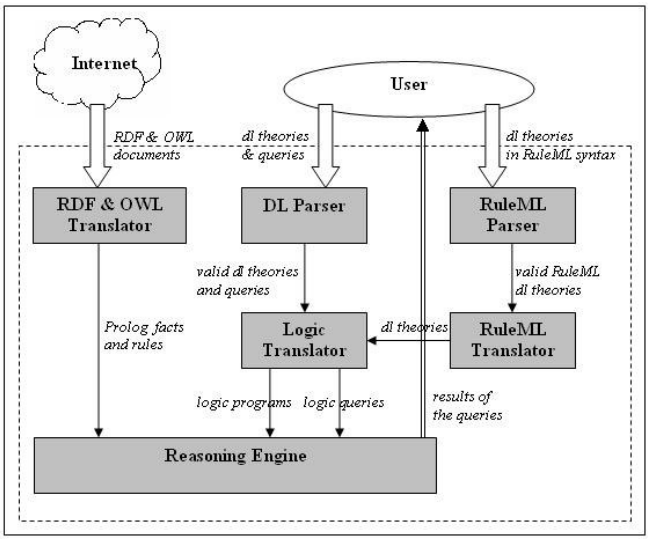

Figure 1: The overall architecture of DR-Prolog

In Figure 1 we present the overall architecture of our system. The system works in the following way: The user imports defeasible theories, either using the syntax of defeasible logic, or in the RuleML syntax, that we describe below in this section. The former theories are checked by the DL Parser, and if they are syntactically correct, they are passed to the Logic Translator, which translates them into logic programs. The RuleML defeasible theories are checked by the RuleML Parser and translated into defeasible theories, which are also passed to the Logic Translator and transformed into logic programs. The Reasoning Engine compiles the logic programs and the metaprogram which corresponds to the user's choice of the defeasible theory variants (ambiguity blocking / propagating), and evaluates the answers to the user's queries. The logic programming system that we use as the Reasoning Engine is XSB. The advantages of this system are two: (a) it supports the well-founded semantics of logic programs through the use of tabled predicates, and its *sk_not* negation operator; and (b) it offers an easy and efficient way to communicate with the other parts of the system. The RDF&OWL Translator is used to translate the RDF/S and OWL information into logical facts and rules, which can be processed by the rules, provided by the user.

The DTD that we have developed to represent defeasible theories in XML format, is in fact an extension of the RuleML DTDs (RuleML). The elements that we add / modify to support the defeasible theories are:

- The "rulebase" root element which uses strict and defeasible rules, fact assertions and superiority relations.
- The "imp" element, which consists of a "_head" and a "_body" element, accepts a "name" attribute, and refers to the strict rules.
- The "def" element, which consists of a "_head" and a "_body" element, accepts a "name" attribute, and refers to the defeasible rules.

- The "superiority" empty element, which accepts the name of two rules as its attributes ("sup" & "inf"), and refers to the superiority relation between these two rules.

Below, we present the modified DTD:

```
<!ELEMENT rulebase ((imp|def|fact|greater)*)>
<!ELEMENT imp ((head, body) | (body, head))>
<!ATTLIST imp
      name ID #IMPLIED>
<!ELEMENT def ((head, body) | (body, head))>
<!ATTLIST def
      name ID #IMPLIED>
<!ELEMENT fact (atom|neg) >
<!ELEMENT greater EMPTY>
<!ATTLIST greater
      sup IDREF #REQUIRED
      inf IDREF #REQUIRED>
<!ELEMENT head (atom|neg)>
<!ELEMENT body (atom|neg)*>
<!ELEMENT neg (atom)>
<!ELEMENT atom ((op,(ind | var)*) | ((ind | var)+, op))>
<!ELEMENT ind (#PCDATA)>
<!ELEMENT var (#PCDATA)>
<!ELEMENT op (#PCDATA)>
```

All the DR-Prolog files are available at:
http://www.csd.uoc.gr/~bikakis/DR-Prolog.

Related Work

There exist several previous implementations of defeasible logics. Conington *et al.* (2002) give the historically first implementation, *D-Prolog*, a Prolog-based implementation. It was not declarative in certain aspects (because it did not use a declarative semantic for the not operator), therefore it did not correspond fully to the abstract definition of the logic. Also, D-Prolog supported only one variation thus it lacked the flexibility of the implementation we report on. Finally it did not provide any means of integration with Semantic Web layers and concepts, a central objective of our work.

Deimos (Maher *et al.*, 2001) is a flexible, query processing system based on Haskell. It implements several variants, but not conflicting literals. Also, it does not integrate with Semantic Web (for example, there is no way to treat RDF data and RDFS/OWL ontologies; nor does it use an XML-based or RDF-based syntax for syntactic interoperability). Thus it is an isolated solution. Finally, it is propositional and does not support variables.

Delores (Maher *et al.*, 2001) is another implementation, which computes all conclusions from a defeasible theory. It is very efficient, exhibiting linear computational complexity. Delores only supports ambiguity blocking propositional defeasible logic; so, it does support ambiguity propagation, nor conflicting literals and variables. Also, it does integrate with other Semantic Web languages and systems, and is thus an isolated solution.

DR-DEVICE (Bassiliades, 2004) is another effort on implementing defeasible reasoning, albeit with a different approach. DR-DEVICE is implemented in Jess, and integrates well with RuleML and RDF. It is a system for query answering. Compared to the work of this paper, DR-DEVICE supports only one variant, ambiguity blocking, thus it does not offer the flexibility of this implementation. At present, it does not support RDFS and OWL ontologies.

SweetJess (Grosof *et al.*, 2002) is another implementation of a defeasible reasoning system (situated courteous logic programs) based on Jess. It integrates well with RuleML. Also, it allows for procedural attachments, a feature not supported by any of the above implementations, not by the system of this paper. However, SweetJess is more limited in flexibility, in that it implements only one reasoning variant (it corresponds to ambiguity blocking defeasible logic). Moreover, it imposes a number of restrictions on the programs it can map on Jess. In comparison, our system implements the full version of defeasible logic.

Conclusion

In this paper we described reasons why conflicts among rules arise naturally on the Semantic Web. To address this problem, we proposed to use defeasible reasoning which is known from the area of knowledge representation. And we reported on the implementation of a system for defeasible reasoning on the Web. It is Prolog-based, supports RuleML syntax, and can reason with monotonic and nonmonotonic rules, RDF facts and RDFS and OWL ontologies..

Planned future work includes:

- Adding arithmetic capabilities to the rule language, and using appropriate constraint solvers in conjunction with logic programs.
- Implementing load/upload functionality in conjunction with an RDF repository, such as RDF Suite (Alexaki *et al*., 2001) and Sesame (Broekstra *et al*., 2003).
- Applications of defeasible reasoning and the developed implementation for brokering, bargaining, automated agent negotiation, and security policies.

References

Alexaki, S.; Christophides, V.; Karvounarakis, G.; Plexousakis, D.; and Trolle, K. 2001 The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. 2nd International Workshop on the Semantic Web (SemWeb'01).

Antoniou, G., and Arief, M. 2002. Executable Declarative Business rules and their use in Electronic Commerce. *In Proc. ACM Symposium on Applied Computing*

Antoniou, G.; Billington, D.; and Maher M. J. 1999. On the analysis of regulations using defeasible rules. *In Proc. 32nd Hawaii International Conference on Systems Science*

Antoniou G.; Billington D.; Governatori G.; and Maher M. J. 2001. Representation results for defeasible logic.

ACM Transactions on Computational Logic 2, 2 (2001): 255 - 287

Antoniou G.; Maher M. J.; and Billington D. 2000a. Defeasible Logic versus Logic Programming without Negation as Failure. *Journal of Logic Programming* 41,1 (2000): 45-57

Antoniou G.; Billington, D.; Governatori G.; and Maher M. J. 2000b: A Flexible Framework for Defeasible Logics. In *Proc. AAAI' 2000*, 405-410

Antoniou G.; Maher M. J. 2002. Embedding Defeasible Logic into Logic Programs. In *Proc. ICLP 2002,* 393-404

Ashri, R.; Payne, T.; Marvin, D; Surridge, M.; and Taylor S. 2004. Towards a Semantic Web Security Infrastructure. In *Proc. of Semantic Web Services 2004 Spring Symposium Series*, Stanford University, California

Bassiliades, N; Antoniou, G; and Vlahavas, I. 2004. DR-DEVICE: A Defeasible Logic System for the Semantic Web. In Proc. 2nd Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR04), LNCS, Springer 2004 (accepted)

Berners-Lee, T; Hendler, J; and Lassila, O. 2001. The Semantic Web. *Scientific American*, 284, 5 (2001): 34-43

Broekstra, J; Kampman, A.; and van Harmelen, F. 2003 Sesame: An Architecture for Storin gand Querying RDF Data and Schema Information. In: D. Fensel, J. A. Hendler, H. Lieberman and W. Wahlster (Eds.), *Spinning the Semantic Web*, MIT Press, 197-222

Connolly, D; van Harmelen, F.; Horrocks, I.; McGuinness, D. L.; Patel-Schneider, P. F.; and Stein, L. A. 2001. *DAML+OIL Reference Description*. www.w3.org/TR/daml+oil-reference

Covington, M. A.; Nute, D.; and Vellino, A. 1997. *Prolog Programming in Depth*, 2nd ed. Prentice-Hall

Dean, M., and Schreiber, G. (Eds.) 2004. *OWL Web Ontology Language Reference*. www.w3.org/TR/2004/REC-owl-ref-20040210/

van Gelder, A.; Ross, K.; and Schlipf, J. 1991. The well-founded semantics for general logic programs. *Journal of the ACM* 38 (1991): 620—650

Governatori, G; Dumas, M.; ter Hofstede, A.; and Oaks, P. 2001. A formal approach to legal negotiation. In *Proc. ICAIL 2001*, 168-177

Grosof, B. N. 1997. Prioritized conflict handing for logic programs. In *Proc. of the 1997 International Symposium on Logic Programming*, 197-211

Grosof, B. N.; Gandhe, M. D.; and Finin T. W. 2002 SweetJess: Translating DAMLRuleML to JESS. RuleML 2002. In: *Proc. International Workshop on Rule Markup Languages for Business Rules on the Semantic Web*

Grosof, B. N.; Horrocks, I.; Volz, R; and Decker, S. 2003. Description Logic Programs: Combining Logic Programs with Description Logic". In: *Proc. 12th Intl. Conf. on the World Wide Web (WWW-2003)*, ACM Press

Grosof, B. N., and Poon, T. C. 2003. SweetDeal: representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In *Proc. 12th International Conference on World Wide Web*. ACM Press, 340 – 349

Levy, A., and Rousset M. C. 1998. Combining Horn rules and description logics in CARIN. *Artificial Intelligence* 104, 1-2 (1998):165 - 209

Li, N.; Grosof, B. N.; and Feigenbaum, J. 2003. Delegation Logic: A Logic-based Approach to Distributed Authorization. In: *ACM Transactions on Information Systems Security* 6,1 (2003)

Maher, M. J. 2002: A Model-Theoretic Semantics for Defeasible Logic. In *Proc. Paraconsistent Computational Logic 2002*, Datalogisker Srkifter 95 ,67-80

Maher, M. J. 2001. Propositional Defeasible Logic has Linear Complexity. *Logic Programming Theory and Practice* 1(6): 691-711 (2001)

Maher, M. J.; Rock, A.; Antoniou, G.; Billington, D.; and Miller, T. 2001. Efficient Defeasible Reasoning Systems. *International Journal of Tools with Artificial Intelligence* 10,4 (2001): 483--501

Marek, V. W., and Truszczynski, M. 1993. *Nonmonotonic Logics; Context Dependent Reasoning*. Springer Verlag

Nute, D. 1994. Defeasible logic. In *Handbook of logic in artificial intelligence and logic programming (vol. 3): nonmonotonic reasoning and uncertain reasoning*. Oxford University Press

RuleML. *The Rule Markup Language Initiative*. www.ruleml.org

SWI. SWI-Prolog, http://www.swi-prolog.org

Wagner, G. 2003. Web Rules Need Two Kinds of Negation. In Proc. First Workshop on Semantic Web Reasoning, LNCS 2901, Springer 2003, 33-50

XSB, Logic Programming and Deductive Database System for Unix and Windows. http://xsb.sourceforge.net