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

Grigoris Antoniou and Antonis Bikakis

Computer Science Department, University of Crete, Greece

Institute of Computer Science, FORTH, Greece

{antoniou,bikakis}@ics.forth.gr

Abstract

Non-monotonic rule systems are expected to play an important role in the layered development of the Semantic Web. Defeasible reasoning is a direction in nonmonotonic reasoning that is based on the use of rules that may be defeated by other rules. It is a simple, but often more efficient approach than other nonmonotonic rule systems, for reasoning with incomplete and inconsistent information. This paper 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.

Keywords: rules, semantic web reasoning, nonmonotonic reasoning

1 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].

On top of the ontology layer sit the logic and proof layers. The implementation of these two layers will allow the user to state any logical principles, and permit the computer to infer new knowledge by applying these principles on the existing data. *Rule systems* can be utilized in two stages of the layered development of the Semantic Web. In the ontology layer, they can serve as extensions to the description logic based ontology languages, by enriching them with more expressive power and representational capabilities. On top of the ontologies, they can be used for the development of automated reasoners that can deduce new knowledge based on the given knowledge. Among the studies regarding rule languages and rule systems for the Semantic Web, we distinguish: (a) the Semantic Web Rules Language [Horrocks *et al.*, 2005], a Horn clause extension of OWL; (b) the study made by Rosati, dealing with reasoning in description logic knowledge bases augmented with rules expressed in Datalog [Rosati, 2005]; (c) the Description Logic Programs and Description Horn Logic presented in [Grosof *et al*., 2003]; and (d) Xcerpt, a rule-based, declarative query and transformation language for XML data [Bry, 2004]. Much work on the same subject is also undergoing in the context of REWERSE, a research program on "Reasoning on the Web" that is funded by the European Commission and Switzerland. Recently, W3C launched the Rule Interchange Format (RIF) Working Group. The group is chartered to produce a language for the exchange of rules and their transfer between rule systems.

Most studies have focused on the employment of monotonic logics in the layered development of the Semantic Web. Nonmonotonic rule systems, on the other hand, seem also to be a good solution, as they offer more expressive capabilities and are closer to commonsense reasoning. Several nonmonotonic logics have been proposed and studied during the last decades, among them *default logic* [Reiter, 1980], *autoepistemic logic* [Moore, 1985], *circumscription* [McCarthy, 1977]. Our work is based on *defeasible reasoning*.

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.

Due to its features, Defeasible Reasoning is appropriate for cases of highly dynamic environments, where the available data is not always complete and unambiguous, or it is continually changing. There is a study comparing various nonmonotonic approaches with human reasoning strategies [Ford and Billington, 2000], in which Defeasible Reasoning performs very well.

Some interesting applications that have so far been developed are: (a) embedded control systems, which use a set of defeasible rules for logically controlling an air-conditioning system [Covington, 1997] and an elevator [Covington, 2000]; (b) the integration of defeasible logic in a decision support system for the legal domain [Johnston and Governatori, 2003]; (c) a stock market agent responsible for gathering stock market information via the Web, and for making decisions according to the stock trading strategies expressed as defeasible theories [Garcia *et al*.,

2000]; and (d) a system for automated agent negotiation, in which the strategies of the negotiating parties are represented in Defeasible Logic [Skylogiannis *et al*., 2001]. The application of defeasible rules for automated negotiation is also studied in [Governatori *et al*., 2000].

The main contribution of this paper is that it presents an implemented defeasible reasoning system (DR-Prolog), which has been tested, evaluated and compared with existing similar implementations. Through the description of the system, we also show how we can combine the expressive power of a nonmonotonic logic (defeasible logic) with the Semantic Web technologies (RDF(S), OWL, RuleML) to build applications for the logic and proof layers of the Semantic Web. The main characteristics of DR-Prolog 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*., 2006] 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 are 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. The latter happens through the transformation of the RDFS constructs and many OWL constructs into rules. Note, however, that a number of OWL constructs cannot be captured by the expressive power of rule languages.

As a result of the above, DR-Prolog is a powerful declarative system supporting (a) rules, facts and ontologies; (b) all major Semantic Web standards: RDF(S), OWL, RuleML; and (c) monotonic and nonmonotonic rules, open and closed world assumption, and reasoning with inconsistencies.

The paper is organized as follows. Section 2 presents the main motivations for nonmonotonic rule systems on the Semantic Web. Section 3 describes the basic ideas of defeasible reasoning. Sections 4 reports on the translation of (a) defeasible theories and (b) RDF, RDFS and (parts of) OWL ontologies into logic programs. Section 5 describes the architecture of the implemented system. Section 6 presents the results of the performance evaluation. Section 7 presents a concrete example of travel packages brokering, showing the functionality of DR-Prolog. Section 8 discusses related work, and Section 9 concludes with a summary and some ideas for future work.

2 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 section 3, 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 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 (e.g. DLP [Grosof *et al*., 2003]) 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.

3 Defeasible Logics

3.1 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
- They have been studied much more deeply, with strong results in terms of proof theory [Antoniou *et al*., 2001], semantics [Governatori *et* al, 2004; 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; Antoniou *et al.*, 2006].

In the following we discuss in more detail some of the ideas and concepts mentioned here.

3.2 Syntax

A *defeasible theory D* is a triple (F ,R ,>) where F 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 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

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

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

that 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.

3.3 Proof Theory

We now give a short informal presentation of how conclusions are drawn in Defeasible Logic. A conclusion P can be derived if there is a rule whose conclusion is P, whose prerequisites (antecedents) are either already been proved or given in the case at hand (i.e. facts), and any stronger rule whose conclusion is in C(P) has prerequisites that fail to be derived. In other words, a conclusion P is (defeasibly) derivable when:

• P is a fact; or

• there is an applicable strict or defeasible rule for P, and either

– all the rules for P-complementary literals are discarded or

– every rule for a P-complementary literal is weaker than an applicable rule for P.

A full definition of the proof theory can be found in [Antoniou *et al.*, 2001]. Roughly, the rules with head P form a team that competes with the team consisting of the rules for P-complementary rules. If the former team wins P is defeasibly provable, whereas if the opposing team wins, P is non-provable.

Governatori *et. al* [2004] describe Defeasible Logic and its variants in argumentation-theoretic terms. A model theoretic semantics is found in [Maher, 2002], and denotational semantics is discussed in [Maher, 2000].

3.4 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: L1,…,Ln, ¬M1,…, ¬Mk ⇒ L
```

can be replaced by the rules:

r: L1,…,Ln, neg(M1),…,neg(Mk) ⇒ L

⇒ neg(M1) M1 ⇒ ¬neg(M1)

… …

⇒ neg(Mk) Mk ⇒ ¬neg(Mk)

where neg(M1),…,neg(Mk) are new auxiliary atoms. If we restrict attention to the original language, the set of conclusions remains the same.

3.5 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 $\neg p$ 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) ⇒ pacifist(X) quaker(a)

r_2: republican(X) ⇒ ¬pacifist(X) republican(a)

r_3: pacifist(X) ⇒ ¬hasGun(X) livesInChicago(a)

r_4: livesInChicago(X) ⇒ hasGun(X) $r_3 > r_4$

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.

3.6 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 theory is propagated to the “new” rules.

4 Translation into Logic Programs

4.1 Translation of Defeasible Theories

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

p is defeasibly provable in $D \Leftrightarrow$

p is included in the Well-Founded Model of $P(D)$

The main reason for the choice of well-founded semantics is its low computational complexity. The connection between defeasible logics and the Stable Model Semantics is thoroughly studied and discussed in [Antoniou *et al.*, 2006].

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

- The translation of [Antoniou *et al.*, 2006; Maher *et al.*, 2001] where a meta-program is 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 the latter approach generates a large number of program clauses. Therefore, we have adopted the meta-program approach.

Translation into logical facts

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,[`$q_1,\ldots,q_n$`]).` for each rule $r: q_1, q_2, \ldots, q_n \rightarrow p \in R$

`defeasible(`r_i`,p,[`$q_1,\ldots,q_n$`]).` 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 $r > s$

Ambiguity Blocking Metaprogram

The metaprogram for the ambiguity blocking version of defeasible logic consists of the following program clauses:

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

```
c1: supportive_rule(Name,Head,Body):-  strict(Name,Head,Body).
c2: supportive_rule(Name,Head,Body):-  defeasible(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.

```
c3: definitely(X):- fact(X).
c4: definitely(X):- strict(R,X,[Y1,…,Yn]), definitely(Y1),…,definitely(Yn).
```

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 the literal 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.

```
c5: defeasibly(X):- definitely(X).
c6: defeasibly(X):- sk_not definitely(~X), supportive_rule(R,X,[Y1,…,Yn]),
                    defeasibly(Y1),…,defeasibly(Yn), sk_not overruled(R,X).
```

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

```
c7: overruled(R,X):- supportive_rule(S,~X, [U1,…,Un]),
                     defeasibly(U1),…,defeasibly(Un),sk_not(defeated(S,~X)).
```

The last clause defines that a rule is defeated when there is a superior conflicting rule, the premises of which are defeasibly provable.

```
c8: defeated(S,X):- sup(T,S),  supportive_rule(T,~X, [U1,…,Un]),
                    defeasibly(U1),…,defeasibly(Un).
```

Ambiguity Propagating Metaprogram

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

```
c7': overruled(R,X):- supportive_rule(S,~X, [U1,…,Un]),
                      supported(U1),…,supported(Un),sk_not(defeated(S,~X)).
```

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.

```
c5: supported(X):- definitely(X).
c6: supported(X):- sk_not definitely(~X), supportive_rule(R,X,[Y1,…,Yn]),
                   supported(Y1),…,supported(Yn), sk_not defeated(R,X).
```

4.2 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 the following 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).
```

All the above rules are created at compile-time, i.e. before the actual querying takes place. Therefore, although at first they seem second-order, because they contain variables in place of predicate names, they are actually first-order rules, i.e. predicate names are constant at run-time.

5 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.

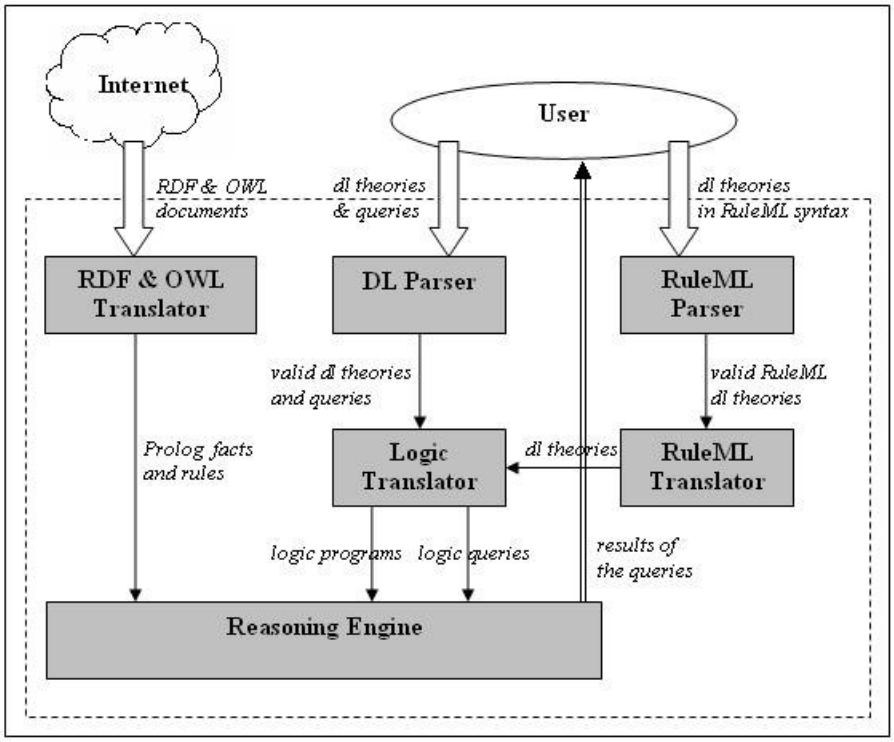

Figure 1: The overall architecture of DR-Prolog

In Figure 1 we present the overall architecture of the 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, which is described below. The former theories are checked by the DL Parser, and if they are syntactically correct, they are passed to the Logic Translator, and are translated into logic programs. The RuleML theories are checked by the RuleML Parser and translated initially into defeasible theories, and then into logic programs. The Reasoning Engine compiles the logic programs and the meta-program which corresponds to DL version that the user selects (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 basically 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.

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" and "def" elements, which consist of a "_head" and a "_body" element, accept a "name" attribute, and refer respectively to the strict and the defeasible rules of the theory.
- 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.

6 Performance Evaluation

In this Section we report on the experimental evaluation that we conducted in order to measure the performance of DR-Prolog, and compare it with the performance of other defeasible reasoning systems, namely *Deimos* [Maher *et al.*, 2001], and *d-Prolog* [Covington *et al.*, 1997]. The basic characteristics of these two systems are presented in Section 8.

6.1 Design of the experiments

We employed the *DTScale* tool of Deimos to create the experimental tests for the evaluation. The tests are defeasible theories, consisting of a large number of facts, rules and superiority relations. The test theories do not contain user-defined conflicting literals, as this feature is not supported by the other systems. In the experiments we focus on defeasible inference, assuming the ambiguity blocking behavior of the test theories (Deimos and d-Prolog do not support the ambiguity propagation). The types of defeasible theories that we created are:

- **Chain theories, chain**(n): They start start with a fact a_o and continue with a chain of defeasible rules of the form $a_{i-1} \Rightarrow a_i$. A variant **chains**(n) uses only strict rules.
- **Circle theories, circle**(n): They consist of n defeasible rules $a_i \Rightarrow a_{(i+1) \bmod n}$. A variant **circles**($n$) uses only strict rules.
- **Levels theories, levels**(n): They consist of a cascade of $2n+2$ disputed conclusions a_i, $i \in [0 \ldots 2n+1]$. For each i, there are rules $\Rightarrow a_i$ and $a_{i+1} \Rightarrow \neg a_i$. For each odd i a priority asserts that the latter rule is superior. A final rule $\Rightarrow a_{2n+2}$ gives uncontested support for a_{2n+2}. A variant **levels**$^-$(n) omits the priorities.
- **Teams theories, teams**(n): They consist of conclusions a_i which are supported by a team of two defeasible rules and attacked by another team of two defeasible rules. Priorities ensure

that each attacking rule is beaten by one of supporting rules. The antecedents of these rules are in turn supported and attacked by cascades of teams of rules.

- **Tree theories, tree**(*n,k*)*:* In tree theories a_0 is at the root of a *k*-branching tree of depth *n* in which every literal occurs once.
- **Directed acyclic graph theories, dag**(*n,k*): In directed acyclic graph theories, a_o is a the root of a *k*-branching tree of depth *n* in which every literal occurs *k* times.

In Table 1, we record the size of the test theories. The reported metrics are: the number of facts in the theory (*facts*); the number of rules in the theory (*rules*); the number of priorities in the theory (*priorities*); the overall "size" of the theory, defined as the sum of the number of facts, rules, priorities and literals in the bodies of all rules.

theory	*facts*	*rules*	*priorities*	*size*
$\mathbf{chain}(n)$	1	n	0	$2n+1$
$\mathbf{chain^s}(n)$	1	n	0	$2n+1$
$\mathbf{circle}(n)$	0	n	0	$2n$
$\mathbf{circle^s}(n)$	0	n	0	$2n$
$\mathbf{levels}(n)$	0	$4n+5$	$n+1$	$7n+8$
$\mathbf{levels^-}(n)$	0	$4n+5$	0	$6n+7$
$\mathbf{teams}(n)$	0	$4\sum_{i=0}^{n} 4^i$	$2\sum_{i=0}^{n} 4^i$	$10\sum_{i=0}^{n-1} 4^i + 6(4^n)$
$\mathbf{tree}(n,k)$	k^n	$\sum_{i=0}^{n-1} k^i$	0	$(k+1)\sum_{i=0}^{n-1} k^i + k^n$
$\mathbf{dag}(n,k)$	k	$nk+1$	0	$nk^2+(n+2)k+1$
$\mathbf{mix}(m,n,k)$	$2mn$	$2m+2mnk$	0	$2m+4mn+4mnk$

Table 1: The size of the test theories

6.2 Configuration for the experiments

All the experiments were performed on an Intel Pentium M 1.3 GHz with 256MB DDR SDRAM machine, running Windows XP with a paging file of 1440MB.

For the evaluation of DR-Prolog we used the 2.6 version of XSB for Windows. For maximum space used by the global (heap) and local (environment) stack of XSB, we invoke XSB with the '-s' command-line option. For the measurement of the execution time, we used the *cputime(-CPU_Time)* predicate, which returns the CPU_Time at the time of the call in seconds. The difference between results of successive calls to this predicate can measure the time spent in specific predicates.

The execution times of Deimos were measured using the –m option of DTScale. We also used the RTS options: -K20M, -M100M. In this way, the system begins with a stack space of 20M and a heap of 100M.

For the compilation of d-Prolog, we used the 5.2.13 version of SWI-Prolog for Windows. The times were measured using the SWI-Prolog *statistics* built-in. When timing several experiments in the same Prolog session, the first experiment consistently took significantly longer than later identical experiments. In our data we have omitted the first timing in a session.

6.3 Experimental Results

Tables 2 and 3 present the time (in CPU seconds) required to find the appropriate conclusion for a_0. The experiments are designed to execute all rules and literals of each test theory.

The times for Deimos include time spent garbage collecting, whereas the times for our system and d-Prolog do not. This adds significantly to the time in problems where the space usage approaches the heap space allocated to the Haskell run-time environment. We must note that the times presented in the tables below do not include the time spent to build and compile the test theories. In the case of our system and d-Prolog, the compilation of the test theories adds a significant amount of time to the overall time of the execution of the experiments. There are also cases that XSB and SWI-Prolog could not compile the test theories, because the default memory

allocation was exhausted. As a result, we could not test our system in cases of theories with more than 20000 logical rules.

	Size	DR-Prolog	Deimos	d-Prolog
chains(n)				
n = 1000	2001	0.29	0.19	0.00
n = 2000	4001	1.08	0.64	0.01
n = 5000	10001	6.58	4.72	0.02
chain(n)				
n = 1000	2001	0.56	0.43	0.13
n = 2000	4001	1.98	1.48	0.25
n = 5000	10001	11.81	8.81	0.62
circles(n)				
n = 1000	2000	0.49	0.16	∞
n = 2000	4000	1.75	0.61	∞
n = 5000	10000	10.57	3.67	∞
circle(n)				
n = 1000	2000	0.48	0.25	∞
n = 2000	4000	1.73	0.90	∞
n = 5000	10000	10.44	5.60	∞
tree(n,k)				
n = 6, k = 3	2185	0.22	0.22	0.06
n = 7, k = 3	6559	1.16	1.37	0.15
n = 8, k = 3	19681	9.61	5.27	0.38
Dag(n,k)				
n = 3, k = 3	43	0.01	0.00	0.06
n = 4, k = 4	89	0.01	0.00	8.80
n = 50, k = 5	1511	0.13	0.06	*
n = 100, k =	11021	2.46	0.49	*

Table 2: Execution times for theories with Undisputed Inferences

In Tables 2, 3 ∞ denotes that the system will not terminate, * denotes that the default memory allocation for XSB or SWI-Prolog was exhausted, - denotes that the experiment was not performed because the runtime required was excessive, ? denotes that the experiment could not be performed. In Table 2 we record the times in the case of theories with undisputed inferences,

namely *chain(n), chains(n), circle(n), circles(n), tree(n,k)* and *dag(n,k)*. In Table 3 we record the times in the case of theories with disputed references, namely *levels(n), levels-(n)* and *teams*.

	Size	DR-Prolog	Deimos	d-Prolog
levels-(n)				
n = 10	67	0.01	0.00	1.61
n = 20	127	0.01	0.01	-
n = 100	607	0.11	0.06	-
n = 1000	6007	3.59	3.53	-
n = 2000	12007	17.96	23.57	-
levels(n)				
n = 10	78	0.01	0.00	1.70
n = 20	148	0.02	0.01	-
n = 100	708	0.09	0.06	-
n = 1000	7008	3.97	3.78	-
n = 2000	14008	19.32	24.06	
teams(n)				
n = 3	594	0.06	0.05	-
n = 4	2386	0.34	0.26	-
n = 5	9554	4.46	1.15	-

Table 3: Execution times for theories with Disputed Inferences

In general, the performance of DR-Prolog is proportional to the size of the problem. This is because the defeasible theories are translated into logical programs with the same number of rules. DR-Prolog performs better in the case of theories that contain strict rules, as in these cases the system has to process a smaller number of rules in order to find the appropriate conclusion.

Comparing to Deimos, DR-Prolog performs a little worse in most of the cases of theories with undisputed inferences. Especially in the case of circle theories, the time that DR-Prolog spends to prove a goal can be more than double of the corresponding time of Deimos. In the case of theories with disputed inferences, the two systems perform almost the same. The only cases that DR-Prolog performs better are the cases of large levels theories. We must say, though, that DR-

Prolog is designed to support rules with variables, while Deimos supports only proportional rules, and this additional feature aggravates the performance of the system.

Comparing to d-Prolog, DR-Prolog performs better in the cases of complex theories (theories with a large number of rules and priorities). Especially in the case of theories with disputed inferences, d-Prolog performs very badly, with time growing exponentially in the problem size. d-Prolog is substantially more efficient than our system when there are only strict rules, due to the direct execution of such rules. However, d-Prolog shows its incompleteness, comparing to the other two systems, when it loops on cyclic theories. We must also note, that d-Prolog spends much more time than DR-Prolog to compile the programs that correspond to the test theories.

We conducted the same experiments for the implementation which is based on the translation, described in [Antoniou and Maher, 2002]. Following this approach, the theories are directly translated into sets of logical rules that the system processes in order to find the appropriate conclusions.

chain(5000)	**circle(5000)**	**tree(8,3)**	**dag(100,10)**	**levels-(1000)**	**levels(1000)**	**teams(5)**
0.10	0.13	0.22	0.11	0.12	0.39	2.3

Table 4: Execution times for the implementation based on the direct translation of the rules

As it is obvious from the results presented in Table 4, by following this implementation the system can succeed much better performance, in terms of execution time. The system performs in this case better than the other two systems in almost all the cases of the test theories.

However, this implementation has two significant drawbacks, comparing to the implementation based on the metaprogram: (a) It does not support queries that contain variables, meaning that it cannot compute at once all the literals that are proved in a defeasible theory; and (b) The translation of defeasible theories results in logical programs, with much bigger size than the corresponding theories, depending on the number of conflicting rules and priorities. Following this

translation, the systems cannot process very large sets of rules, and has to spend too much time compiling the logical programs.

7 A Concrete Example of Travel Packages Brokering

7.1 Used Ontologies and Data

In this section, we present a concrete example in order to show the way that DR-Prolog works and interacts with the user's queries. We define an ontology in RDFS to model concepts and their relationships in the domain of travel industry. Part of the ontology is depicted in Fig.2.

```
<?xml version="1.0" ?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<rdfs:Class rdf:ID="Itinerary">
  <rdfs:comment>The class of travel packages offered by travel agents</rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="service">
  <rdfs:comment>The class of services included into a travel package</rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="hotel">
  <rdfs:comment>The class of  hotels.</rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="fivestarHotel">
  <rdfs:comment>The class of 5-star hotels.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#hotel"/>
</rdfs:Class>

...
<rdf:Property rdf:ID="includesService">
  <rdfs:comment>It relates a service to a particular itinerary</rdfs:comment>
  <rdfs:domain rdf:resource="#itinerary"/>
  <rdfs:range rdf:resource="#service"/>
</rdf:Property>
<rdf:Property rdf:ID="parking">
  <rdfs:comment>A boolean indicator for the existence of parking</rdfs:comment>
  <rdfs:domain rdf:resource="#hotel"/>
  <rdfs:range rdf:resource="&xsd;boolean"/>
</rdf:Property>
</rdf:RDF>
```

Fig. 2 Part of the Tourism Domain Ontology

We also define a few initial instances regarding descriptions about hotels, islands, travel agents, means of transport, airline companies etc. (example in Fig. 3). The terms which are used for the definition of the instances are reference to the RDFS ontology.

```
<?xml version="1.0"?>
 <!DOCTYPE rdf:RDF [
 <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
 <!ENTITY schema "http://www.csd.uoc.gr/~dogjohn/ontology/TravelOntology.rdfs#">
 <!ENTITY inst "http://www.csd.uoc.gr/~dogjohn/data/TravelOntologyInstances.rdf#">]>
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:instances="&inst;"
     xmlns:schema="&schema;">
    <rdf:Description rdf:about="CretaMareRoyal">
     <rdf:type rdf:resource="&schema;fivestarHotel" />
     <schema:resortID rdf:datatype="&xsd;string">1</schema:resortID>
     <schema:hotelName rdf:datatype="&xsd;string">Creta Mare Royal</schema:hotelName>
     <schema:hotelCategory rdf:datatype="&xsd;string">Business</schema:hotelCategory>
     <schema:parking rdf:datatype="&xsd;boolean">true</schema:parking>
     <schema:swimmingPool rdf:datatype="&xsd;boolean">true</schema:swimmingPool>
     <schema:breakfast rdf:datatype="&xsd;boolean">true</schema:breakfast>
     <schema:distanceFromSea rdf:datatype="&xsd;integer">10</schema:distanceFromSea>
    </rdf:Description>
</rdf:RDF>
```

Fig. 3 Part of the Travel Ontology Instances

```
<?xml version="1.0"?>
 <!DOCTYPE rdf:RDF [
 <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
 <!ENTITY schema "http://www.csd.uoc.gr/~dogjohn/ontology/TravelOntology.rdfs#">
 <!ENTITY inst "http://www.csd.uoc.gr/~dogjohn/data/TravelOntologyInstances.rdf#">]>
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:instances="&inst;"
     xmlns:schema="&schema;">
    <rdf:Description rdf:about="IT1">
     <rdf:type rdf:resource="&schema;itinerary"/>
     <schema:from rdf:datatype="&xsd;string">athens</schema:from>
     <schema:to rdf:datatype="&xsd;string">Creta</schema:to>
     <schema:departureDate rdf:datatype="&xsd;string">3/8/2004</schema:departureDate>
     <schema:returnDate rdf:datatype="&xsd;string">15/8/2004</schema:returnDate>
     <schema:persons rdf:datatype="&xsd;integer">2</schema:persons>
     <schema:price rdf:datatype="&xsd;integer">1000</schema:price>
     <schema:offeredBy   rdf:resource="&inst;ZORP" />
     <schema:includesResort  rdf:resource="&inst;CretaMareRoyal" />
      <schema:includesTransportation rdf:resource="&inst;Minoan" />
     <schema:includesService rdf:resource="&inst;AVIS" />
     <schema:forPlace rdf:resource="&inst;Creta" />
    </rdf:Description>
</rdf:RDF>
```

Figure 4. Expression of an advertisement in RDF

7.2 Expression of Offers

Every travel agency, which is a potential service provider, can publish an offer to the broker. After publication, the offer is considered as an advertisement. The advertisement regards a complete travel package and its format is depicted in Fig. 4. The advertisement of this example is submitted by the travel agency "Zorpidis S.A" and regards a travel package for two persons for the island of Crete. It includes a hotel, local transportation service (a car) and tickets for the ferry, and costs 1000 money units.

7.3 Formalization of Requirements and Preferences

On the other hand, every customer who is a potential service requester can express his requirements and preferences to the broker. Consider the following example: "Teo, a busy businessman, has the following preferences about his holiday travel package: First of all, he wants to depart from Athens and he considers that the hotel at the place of vacation must offer breakfast. In addition, he would like either the existence of a swimming pool at the hotel to relax all the day, or a car equipped with A/C, to make daily excursions at the island. However, Teo believes that if there is no parking area at the hotel, the car is useless, because he adds to him extra effort and fatigue. Lastly, if the tickets for his transportation to the island are not included in the travel package, he is not willing to accept it…"

$\mathbf{r_1}$: itinerary(X),from(X,athens),includesResort(X,Y),hotel(Y),
breakfast(Y,true) $\Rightarrow$ accept(X)
$\mathbf{r_2}$: itinerary(X),from(X,athens),includesResort(X,Y),hotel(Y),
swimmingPool(Y,true) $\Rightarrow$ accept(X)
$\mathbf{r_3}$: itinerary(X),from(X,athens),includesService(X,Z),hasVehicle(Z,W),
vehicleAC(W,true) $\Rightarrow$ accept(X)
$\mathbf{r_4}$: itinerary(X),includesResort(X,Y),hotel(Y),parking(Y,false)
$\Rightarrow$ ~accept(X)
$\mathbf{r_5}$: itinerary(X),~includesTransportation(X,Z) $\Rightarrow$ ~accept(X)

$r_4 > r_3,\ r_1 > r_4,\ r_2 > r_4,\ r_5 > r_1,\ r_5 > r_2,\ r_5 > r_3$

Figure 5. Expression of User's Requirements in Defeasible Logic

This verbal description of Teo's requirements about acceptable offers can be modeled through the following rules, depicted in Fig. 5. More rules and priorities could be used to express selection preferences among acceptable offerings.

The id of the travel package described in Figure 4 will be returned as an answer, as its specifications comply with the rules described in Figure 5. To perform the reasoning, DR-Prolog translates the RDF / RDFS descriptions of the available data (travel packages, hotels, etc.) into logical facts, and the rules describing the user's preferences into logical facts and rules; it then uses the metaprogram (described in section 4) and the rules capturing the semantics of RDF Schema to reason with the rules and the ontology data. Among the user's rules, r_1 will fire when 'X' takes the value "IT1" (the id of the travel package described in Figure 4) and 'Y' takes the value 'CretaMareRoyal" (the id of the hotel included in that package). Although 'CretaMareRoyal' is described as an instance of the 5-star hotels' class, it can also be regarded as an instance of the class "hotels" using the rdfs schema information that 5-star hotels are a subclass of the class "hotels" (Figure 2) and the rule describing the rdfs:subclass relation (section 4). The only rule that can override r_1, r_5, will not fire as for 'IT1' the includesTransportation property is evaluated to true.

8 Related Work

There exist several previous implementations of defeasible logics. Conington *et al.* [1997] 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. In addition, DR-Prolog uses Logic Programs with Well Founded Semantics, which is formally equivalent to the formal model described in section 3. In contrast, DR-DEVICE uses the logic metaprogram as a guiding principle, but there is no formal proof of the correctness of the implementation. On the other hand, DR-DEVICE has the relative advantage of easier integration with mainstream software technologies.

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.

Apart from relating DR-Prolog to previous implementations of similar logical systems, there is an interesting theoretical question concerning the relationship of defeasible logics with preferential logics [Shoham, 1987]. Preferential logics provide a framework in which a monotonic logic is augmented by a preference relation on models, thus supporting both monotonic and nonmonotonic conclusions; for defeasible logics their counterpart would be strict and defeasible conclusions. The model theoretic semantic of defeasible logic [Maher 2002] may be a starting point for this investigation. Obstacles to be addressed include the different target of preference in the two approaches (rules versus models) and required properties of the preference relation (in preferential logics it is usually a partial order or a strict partial order, while defeasible logics just require the preference relation to be acyclic).

9 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 that is known from the area of knowledge representation, and we reported on the implementation of a system for defeasible reasoning on the Web. The proposed system 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, mobile computing and security policies.

References

[Alexaki *et al.*, 2001] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle. The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. In *Proc. 2nd International Workshop on the Semantic Web*, Hongkong, May 1, 2001.

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

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

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

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

[Antoniou *et* al., 2001] G. Antoniou, D. Billington, G. Governatori, and M.J. Maher. 'Representation results for defeasible logic'. *ACM Transactions on Computational Logic* 2, 2 (2001): 255 - 287

[Antoniou *et* al., 2006] G. Antoniou, D. Billington, G. Governatori, and M. Maher. 'Embedding Defeasible Logic into Logic Programming', *Theory and Practice of Logic Programming*, to appear

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

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

[Bassiliades *et al*., 2004] N. Bassiliades, G. Antoniou, and I. Vlahavas (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 *et al*., 2001] T. Berners-Lee, J. Hendler, and O. Lassila. 'The Semantic Web'. *Scientific American*, 284, 5 (2001): 34-43

[Broekstra *et al*., 2003] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: An Architecture for Storing and 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, 2003

[Bry, 2004] F. Bry and S. Schaffert: Querying the Web Reconsidered: A Practical Introduction to Xcerpt. *Extreme Markup Languages*, 2004

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

[Covington, 1997] M. Covington. Defeasible Logic on an Embedded Microcontroller. *Proceedings, Tenth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA-AIE)*, 1997.

[Covington, 2000] M. Covington. 'Logical control of an elevator with defeasible logic'. *IEEE Transactions on Automatic Control*, 45(7):1347--1349, 2000

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

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

[Ford and Billington, 2000] M. Ford and D. Billington. 'Strategies in human nonmonotonic reasoning'. *Computational Intelligence* 16,3(2000):446 -468.

[Garcia *et* al., 2000] A. Garcia, D. Gollapally, P. Tarau, and G. Simari. Deliberative stock market agents using Jinni and defeasible logic programming. In *Proc. of the ECAI Workshop on Engineering Societies in the Agents' World,* Berlin, Germany, August 2000

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

[Governatori *et al*., 2000] G. Governatori, A.H.M. ter Hofstede, and P. Oaks. Defeasible Logic for Automated Negotiation, P.M. Swatman and P. Swatman (Eds.), *Proceedings of ColleCTeR*, Deakin University, 2000

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

[Governatori *et al*., 2004] G. Governatori, M. J. Maher, G. Antoniou, and D. Billington. 'Argumentation Semantics for Defeasible Logics'. *Journal of Logic and Computation* 14,5 (2004): 675-702

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

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

[Grosof *et al*., 2003] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic Programs: Combining Logic Programs with Description Logic". In: *Proc. 12th Intl. Conf. on the World Wide Web*, ACM Press, 2003

[Grosof and Poon, 2003] B. N. Grosof and T. C. Poon. 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, 2003.

[Horrocks *et al.*, 2005] I. Horrocks, P. F. Patel-Schneider, S. Bechhofer, and D. Tsarkov. 'OWL Rules: A Proposal and Prototype Implementation'. *Journal of Web Semantics*, Vol. 3, No. 1, pp 23-40, 2005.

[Johnston and Governatori, 2003] B. Johnston and G. Governatori. Induction of Defeasible Logic Theories in the Legal Domain. In Sartor, Giovanni, Eds. *Proceedings of the 9th International Conference on Artificial Intelligence and Law (ICAIL-03)*, 24--28, pages 204-213, Edinburgh, Scotland, June, 2003.

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

[Maher, 2000] M. J. Maher. A Denotational Semantics for Defeasible Logic, *Proc. First International Conference on Computational Logic*, LNAI 1861, Springer, 209-222, 2000.

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

[Maher *et al*., 2001] M. J. Maher, A. Rock, G. Antoniou, D. Billington and T. Miller. 'Efficient Defeasible Reasoning Systems'. *International Journal of Tools with Artificial Intelligence* 10,4 (2001): 483—501

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

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

[McCarthy, 1977] J. McCarthy. Epistemological problems of artificial intelligence. In *Proceedings of the International Conference on Artificial Intelligence*, pages 223-227, Cambridge, MA, 1977.

[Moore, 1985] R. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*,25,1(1985):75-94

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

[Reiter, 1980] R. Reiter. 'A Logic for Default Reasoning'. *Artificial Intelligence* 13(1980):81-132, 1980.

[Rosati, 2005] R. Rosati, 'On the decidability and complexity of integrating ontologies and rules'. *Journal of Web Semantics*, 3, 1 (2005): 41-60.

[RuleML] RuleML. *The Rule Markup Language Initiative*. www.ruleml.org, 2006.

[Shoham, 1987] Y. Shoham. Nonmonotonic logics: Meaning and utility. In J. Mc-Dermott, editor, *Proceedings 10th International Joint Conference on Artificial Intelligence*, pages 388-392. Morgan Kaufmann, 1987.

[Skylogiannis *et al*., 2001] T. Skylogiannis, G. Antoniou, N. Bassiliades, and G. Governatori**,** DR-NEGOTIATE: A System for Automated Agent Negotiation with Defeasible Logic-Based Strategies. In *Proc. 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE-05)*, 2005.

[SWI] SWI-Prolog, http://www.swi-prolog.org, 2006.

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

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