A Defeasible Logic Programming System for the Web

Grigoris Antoniou and Antonis Bikakis

Computer Science Department, University of Crete, Greece

Institute of Computer Science, FORTH, Greece

{ga,bikakis}@csd.uoc.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 and priorities; (c) is based on a translation to logic programming with declarative semantics; and (d) is flexible and adaptable to different intuitions within defeasible reasoning.

1 Introduction

The development of the Semantic Web [9] 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 [11] and OWL [13].

The next step in the development of the Semantic Web will be the logic and proof layers, and *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 (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 [18]. Based on that work and on previous work on hybrid reasoning [20] 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 section 2. 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 [24, 5] and Courteous Logic Programs [16]. 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 [25], the main standardization effort for rules on the Semantic Web.

- It is based on Prolog. The core of the system consists of a translation of defeasible knowledge into Prolog. However, the implementation is declarative because it interprets the not operator using Well-Founded Semantics [14].
- The main focus was 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 paper is organized as follows. Section 2 describes the main motivations for conflicting rules on the Semantic Web. Section 3 describes the basic ideas of default reasoning, and sections 4 and 5 its translations into logic programs and XML files, respectively. Section 6 reports on the implemented system. Section 7 discusses related work, and section 8 concludes with a summary and some ideas for future work.

2 Motivation for Conflicting Rules on the Semantic Web

Reasoning with Incomplete Information: [3] describes 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 [23].

Rules with Exceptions: Rules with exceptions are a natural representation for policies and business rules [4]. And priority information is often implicitly or explicitly available to resolve conflicts among rules. Potential applications include security policies [8, 21], business rules [3], personalization, brokering, bargaining, and automated agent negotiations [15].

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. [19] presents 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 [18]) 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 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

- Defeasible logics are rule-based, without disjunction
- Classical negation is used in the heads and bodies of rules, but negation-as-failure is not used in the object language (it can easily be simulated, if necessary [4])
- Rules may support conflicting conclusions
- The logics are skeptical in the sense that conflicting rules do not fire. Thus consistency is preserved
- Priorities on rules may be used to resolve some conflicts among rules
- The logics take a pragmatic view and have low computational complexity

3.2 Syntax

A *defeasible theory D* is a couple $(R,>)$ where 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) => tenured(X)

r': visiting(X) => ¬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 is not tenured.

A formal definition of the proof theory is found in [5]. A model theoretic semantics is found in [22].

3.3 Ambiguity Blocking and Ambiguity Propagation 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)

r_2: republican(X) => ¬pacifist(X)

r_3: pacifist(X) => ¬hasGun(X)

r_4: livesInChicago(X) => hasGun(X)

$r_3 > r_4$

quaker(a)

republican(a)

livesInChicago(a)

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.

A preference for ambiguity blocking or ambiguity propagating behavior is one of the properties of nonmonotonic inheritance nets over which intuitions can clash [26]. 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.

3.4 Conflicting Literals

So far only conflicts among rules with complementary heads were deected and used. We considered all rules with head *L* as *supportive* of *L*, and all rules with head ¬*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 take may be classified in one of the categories low, medium, and high. The way to solve this problem is to use constraint rules of the form

conflict :: low, medium

conflict :: low, high

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

The superiority relation among the rules of the defeasible theory is propagated to the "new" rules. For example, if the defeasible theory includes the following two rules and a superiority relation among them:

r_1: $q_1,q_2,\ldots,q_n \rightarrow L$

r_2: $p_1,p_2,\ldots,p_n \rightarrow M$

$r_1 > r_2$

we will augment the defeasible theory by :

r_1': $q_1,q_2,\ldots,q_n \rightarrow \neg M$

r_2': $p_1,p_2,\ldots,p_n \rightarrow \neg L$

$r_1 > r_2$'

r_1' $> r_2$

4 Translation into Logic Programs

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 all stable models of *P(D)*

In order to achieve this goal, we based our translation on the translation which makes use of control literals, presented in [7]. We have made some extensions to support superiority relations among rules, and to support both ambiguity blocking and ambiguity propagation behavior. The translation has two versions: the ambiguity blocking version and the ambiguity propagation version.

4.1 Translation of Ambiguity Blocking Behavior

Given a fact p we translate it into the program clause

$a(p)$: *definitely(p).*

Given a strict rule

r: $q_1,q_2,\ldots,q_n \rightarrow p$

we translate it into the program clause

$b(r)$: *definitely(p):- definitely(q_1),definitely(q_2),…,definitely(q_n).*

Additionally, we introduce the clause

$c(p)$: *defeasibly(p):- definitely(p).*

for every literal p. This last clause corresponds to the condition of the defeasible theory: a literal p is defeasibly provable if it is strictly (definitely) provable.

Given a defeasible rule

r: $q_1,q_2,\ldots,q_n \Rightarrow p$

we translate it into the following set of clauses:

$d_1(r)$: *defeasibly(p):- defeasibly(q_1),defeasibly(q_2),…,defeasibly(q_n),*
not[1] *definitely(¬p),ok(r,p).*

$d_2(r)$: *ok(r,x):- ok'(r,s_1),…,ok'(r,s_m).*

where $\{s_1,\ldots,s_m\}$ = {the set of defeasible rules with head: $\neg p$}

$d_3(r,s_i)$: *ok'(r,s_i):- blocked(s_i).* for all $s_i \in \{s_1,\ldots,s_m\}$

$d_4(r,s_i)$: *ok'(r,s_i):- defeated(s_i).* for all $s_i \in \{s_1,\ldots,s_m\}$

$d_5(r,q_i)$: *blocked(r):- not defeasibly(q_i).* for all $i \in \{1,2,\ldots,n\}$

$d_6(r,s_i)$: *defeated(r):- not blocked(s_i), sup(s_i,r).* for all $s_i \in \{s_1,\ldots,s_m\}$

[1] For the implementation of the translation, we use *tnot* as the nagation operator. The use of this operator is described in section 6.

Given a superiority relation

$r > s$

we translate it into the program clause

e(r,s): sup(r,s).

- $d_1(r)$ says that to prove p defeasibly by applying r, we must prove all the antecedents of r, the negation of p should not be strictly (definitely) provable, and it must be ok to apply r.
- $d_2(r)$ says when it is ok to apply a rule r with head p: we must check that it is ok to apply r w.r.t. every rule with head $\neg p$.
- $d_3(r,s_i)$ says that it is ok to apply r w.r.t. s_i is blocked.
- $d_4(r,s_i)$ says that it is ok to apply r w.r.t. s_i is blocked.
- $d_5(r,q_i)$ specifies the only way a rule can be blocked: it must be impossible to prove one of its antecedents.
- $d_6(r,s_i)$ specifies the only way a rule r can be defeated: there must be at least one rule s with complementary head (conflicting rule), which is not blocked and is superior to r.

For a defeasible theory with ambiguity blocking behavior D we define $P(D)$ to be the union of all clauses $a(p)$, $b(r)$, $c(p)$, $d_1(r)$, $d_2(r)$, $d_3(r,s_i)$, $d_4(r,s_i)$, $d_5(r,q_i)$, $d_6(r,s_i)$, $e(r,s)$.

4.2 Translation of Ambiguity Propagation Behavior

We must make some changes to the procedure of the translation that we described above to support ambiguity propagation behavior. Our goal is to ensure that the ambiguity of a conclusion is propagated to its dependents. To achieve this we must define a new predicate: *supported*.

The program clauses $a(p)$, $b(r)$, $c(p)$ remain unchanged. In this version we add a new program clause $s(p)$:

s(p): supported(p):- definitely(p).

for every literal p. This clause says that p is supported if it is strictly (definitely) provable.

The program clauses $d_1(r)$, $d_2(r)$, $d_4(r,s_i)$, $d_5(r,q_i)$, $d_6(r,s_i)$, $e(r,s)$ also remain the same. In order to support the ambiguity propagation behavior, we must change $d_3(r,s_i)$ and add two more program clauses for the defeasible rules. So, given a defeasible rule

$r: q_1,q_2,\ldots,q_n \Rightarrow p$

we translate it into the following set of clauses:

$d_1(r)$, $d_2(r)$,

$d_3'(r,s_i)$: $ok'(r,s_i)$:- $obstructed(s_i)$. for all $s_i \in \{s_1,\ldots,s_m\}$

$d_4(r,s_i)$, $d_5(r,q_i)$, $d_6(r,s_i)$,

$d_7(r,q_i)$: $obstructed(r)$:- $not\ supported(q_i)$. for all $i \in \{1,2,\ldots,n\}$,

$d_8(r)$: $supported(p)$:- $supported(q_1),\ldots,supported(q_n)$, $not\ defeated(r)$.

- $d_3'(r,s_i)$ says that it is ok to apply r w.r.t. s_i is obstructed.
- $d_7(r,q_i)$ specifies the only way a rule can be obstructed: at least one of its antecedents must not be supported.
- $d_8(r)$ says that p is supported by applying r, if all the antecedents of r are supported, and r is not defeated.

For a defeasible theory with ambiguity propagation behavior D we define $P(D)$ to be the union of all clauses $a(p)$, $b(r)$, $c(p)$, $d_1(r)$, $d_2(r)$, $d_3'(r,s_i)$, $d_4(r,s_i)$, $d_5(r,q_i)$, $d_6(r,s_i)$, $d_7(r,q_i)$, $d_8(r)$, $e(r,s)$.

5 Translation into XML files

Another interesting part of our work was the creation of a DTD which would allow us to translate defeasible theories into XML files. This DTD is in fact an extension of the RuleML DTDs [25]. It covers both strict and defeasible rules, as well as the superiority relations between these rules. The elements of the RuleML DTD that we added / modified are:

- The “rulebase” root element which uses “imp” (strict) and “def” (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 of a theory.
- The “def” element which consists of a “_head” and a “_body” element, accepts a “name” attribute, and refers to the defeasible rules of a theory.
- The “superiority” empty element, which accepts the name of two rules as its attributes (“sup” & “inf”), and refers to the superity relation of these two rules.

Below we present the modified DTD:

```
<!ELEMENT rulebase ((imp|def|fact|superiority)*)>
<!ELEMENT imp ((_head, _body) | (_body, _head))>
<!ATTLIST imp
          name ID #IMPLIED>
<!ELEMENT def((_head, _body) | (_body, _head))>
<!ATTLIST def
          name ID #IMPLIED>
<!ELEMENT fact (_head) >
<!ELEMENT superiority EMPTY>
<!ATTLIST superiority
          sup IDREF #REQUIRED
          inf IDREF #REQUIRED>
<!ELEMENT _head (atom)>
<!ELEMENT _body (atom | and)>
<!ELEMENT and (atom*)>
<!ELEMENT atom ((_opr,(ind | var)*) | ((ind | var)+, _opr))>
<!ELEMENT _opr (rel)>
<!ELEMENT ind  (#PCDATA)>
<!ELEMENT var  (#PCDATA)>
<!ELEMENT rel  (#PCDATA)>
```

6 Implementation

Our goal was to develop a system that supports not only the basics of defeasible logic, but also the two different behaviors (ambiguity blocking and ambiguity propagation) of this logic, and the use of conflicting literals. The system consists of five different tools: the *parser*, the *logic translator*, the *XML translator*, the *logic compiler*, and the *evaluator*. We employed *lex & yacc*, to create the parser and the two translators. We use XSB [28] as the logic compiler. The same system is responsible for evaluating the user's queries.

The system can be used either to translate a defeasible theory into an XML file, according to the DTD we described in section 5, or as a query evaluator. The queries that the user can make are of the form: "Can you conclude that the literal *p* of my defeasible theory *D* is / is not proved strictly / defeasibly?". The system can evaluate the answer of one query of this form at a time. It has not the ability to evaluate queries of the form: "Which literals of my defeasibly theory *D* are proved strictly / defeasibly?". The overall procedure is described in Fig.1.

In the follwing sections, we will describe the role of each of the tools that compose the architecture of the system.

6.1 The parser

The parser is responsible for parsing the user's defeasible theory and for checking for the validity of this theory. The theory is considered to be valid, if it follows the standard syntax of defeasible logic, as described in section 3. If there are syntax errors in a defeasible theory, the system informs the user about these errors, and does not proceed to the translation of the theory. If the theory is valid, the parser creates a symbol table, which includes all the facts, rules and superiority relations of the user's defeasible theory. The symbol table will be later used by the translator.

Another important task of the parser is to check for the conflicting literals of the defeasible theory, and to augment the theory with the appropriate rules and superiority relations. If the user has defined two or more literals to be conflicting, the parser checks for the rules which have one of these literals as

their head, and for the superiority relations among these rules, and creates new rules and superiority relations, following the way we described in Section 3.

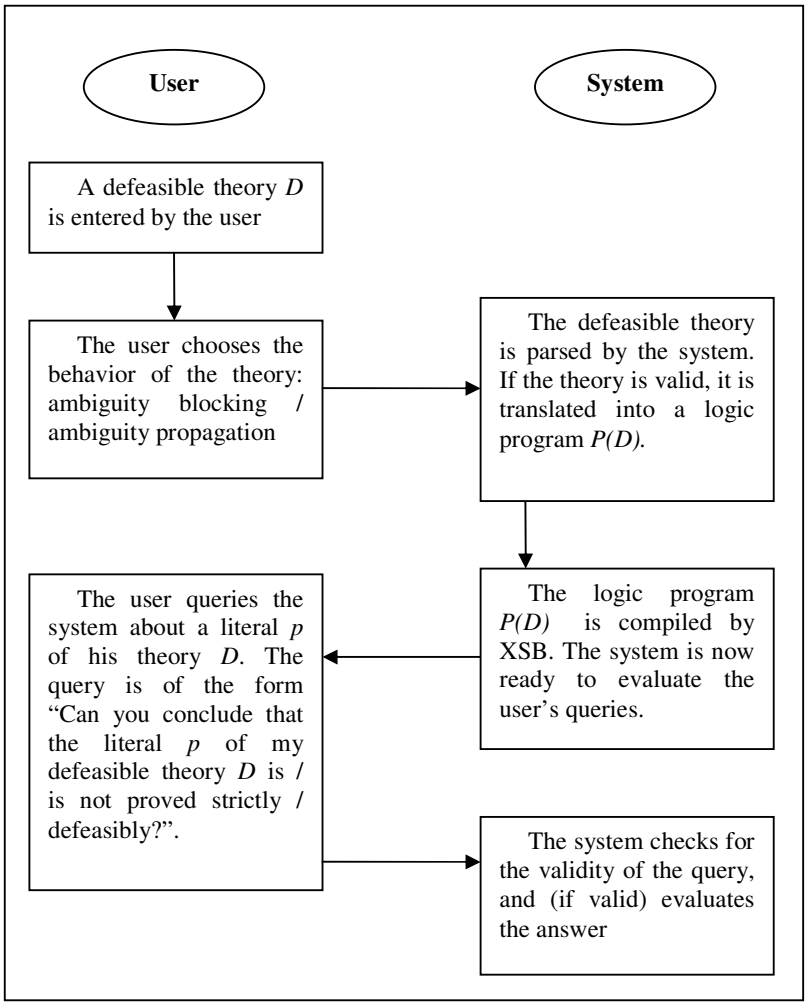

Fig. 1. The Interaction between the system and its users.

The last task of the parser is to check for the validity of the user's queries. We have defined a standard syntax for these queries:

- $+D\ p$: is it concluded that literal p of the defeasible theory is proved strictly?
- $-D\ p$: is it concluded that literal p of the defeasible theory is not proved strictly?
- $+d\ p$: is it concluded that literal p of the defeasible theory is proved defeasibly?
- $-d\ p$:is it concluded that literal p of the defeasible theory is not proved defeasibly?

The syntax we use for the complementary of a literal *p* is ~*p*.

6.2 The logic translator

If the defeasible theory has been parsed with success, the translator creates the logic program which corresponds to the user's defeasible theory. The translator has two inputs and one output. The first input is the user's defeasible theory *D* (checked and possibly augmented with new rules and superiority relations by the parser). The second input is the user's choice of the behavior of the defeasible theory: ambiguity blocking / ambiguity propagation. The output is a logic program *P(D)*, which is in fact a Prolog file. The translation of each defeasible rule to the corresponding Prolog rule is described in section 4. The only difference is that, instead of *not* we use *tnot*, which is XSB's negation operator and allows for the correct execution of programs with well founded semantics. The translator parses the symbol table, which is created by the parser, and translates the defeasible rules one by one. In the course of this procedure, some searches of the symbol table are required. For example, if a translator meets a defeasible rule with head *p*, it searches the symbol table for defeasible rules with complementary head, ~*p*.

The translator is also responsible for transforming the user's queries into valid Prolog queries:

- +D p is translated into definitely(p).
- -D p is translated into not definitely(p).
- +d p is translated into defeasibly(p).
- -d p is translated into not defeasibly(p).

6.3 The XML translator

The role of the XML translator is to translate the defeasible theory, which has already been checked for its validity by the parser, into a valid XML file. A valid defeasible theory acts as input of the translation. The output is an XML file, which is created according to the DTD that we described in section 5.

6.4 The logic program compiler

The logic program compiler employs XSB to compile the logic program *P(D)*, created by the logic translator. We use XSB, as we need a powerful Prolog system for our needs. A defeasible theory which consists of *n* number of facts, rules and superiority relations, is translated into a logic program with *r*n* Prolog rules, where *2<n<6* in the case of ambiguity blocking behavior, and *3<n<8* in the case of ambiguity propagation behavior.

XSB is appropriate for building integrated real-world systems, as it is easy to construct the communication module between XSB and the other parts of such systems. In our case, it was critical for the performance of the system, to find an easy and efficient way to communicate the logic program compiler with the parser and the translator. Only a small number of code was enough to construct this communication module.

6.5 The evaluator

The role of the evaluator is to evaluate the answer to the user's queries. The queries are parsed by the parser, and translated into Prolog queries by the logic translator, before being passed to the evaluator. The Prolog queries are applied to the compiled Prolog file, and a positive ("yes") or a negative answer ("no") is produced by the evaluator.

6. Related Work

There exist several previous implementations of defeasible logics. [12] gives 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.

Deimos [22] 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; nor does it use an XML-based or RDF-based syntax). Thus it is an isolated solution. Finally, it is propositional and does not support variables.

Delores [22] is another implementation, which computes all conclusions from a defeasible theory (the only system of its kind known to us). 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.

RD-DEVICE [27] is another effort on implementing defeasible reasoning, albeit with a different approach. RD-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, RD-DEVICE supports only one variant, ambiguity blocking, thus it does not offer the flexibility of this implementation.

SweetJess [17] 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.

7. 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, and supports RuleML syntax.

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 [1] and Sesame [10].
- Study in more detail integration of defeasible reasoning with description logic based ontologies. Starting point of this investigation will be the Horn definable part of OWL [18].
- Applications of defeasible reasoning and the developed implementation for brokering, bargaining, automated agent negotiation, and personalization.

References

1. S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis and K. Tolle (2001). The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. In *Proc. 2nd International Workshop on the Semantic Web*, Hongkong, May 1, 2001.
2. G. Antoniou (2002). Nonmonotonic Rule Systems on Top of Ontology Layers. In *Proc. 1st International Semantic Web Conference.* Springer, LNCS 2342, 394-398
3. G. Antoniou and M. Arief (2002). Executable Declarative Business rules and their use in Electronic Commerce. In *Proc. ACM Symposium on Applied Computing*
4. G. Antoniou, D. Billington and M.J. Maher (1999). On the analysis of regulations using defeasible rules. In *Proc. 32nd Hawaii International Conference on Systems Science*
5. G. Antoniou, D. Billington, G. Governatori and M.J. Maher (2001). Representation results for defeasible logic. ACM Transactions on Computational Logic 2, 2 (2001): 255 - 287
6. G. Antoniou, M. J. Maher and D. Billington (2000). Defeasible Logic versus Logic Programming without Negation as Failure. *Journal of Logic Programming* 41,1 (2000): 45-57
7. G. Antoniou, M.J. Maher (2002). Embedding Defeasible Logic into Logic Programs. In *Proc. ICLP 2002,* 393-404
8. R. Ashri, T. Payne, D. Marvin, M. Surridge and S. Taylor (2004). Towards a Semantic Web Security Infrastructure. In *Proc. of Semantic Web Services 2004 Spring Symposium Series*, Stanford University, California

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

10. J. Broekstra, A. Kampman and F. van Harmelen (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

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

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

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

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

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

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

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

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

19. B. N. Grosof and T. C. Poon (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

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

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

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

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

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

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

26.D.D. Touretzky, J.F. Horty and R.H. Thomason. (1987). A Clash of Intuitions: The Current State of Nonmonotonic Inheritence Systems. In *Proc. IJCAI-87,* 476-482, Morgan Kaufmann, 1987.

27.N. Vassiliades, G. Antoniou and Y. Vlahavas (2004). RD-DEVICE: A Defeasible Reasoning Systems for the Web. Submitted.

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

29.Lex & Yacc, http://dinosaur.compilertools.net