ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΑΝΑΠΤΥΞΗ ΣΥΣΤΗΜΑΤΟΣ ΑΝΑΙΡΕΣΙΜΩΝ ΚΑΝΟΝΩΝ ΓΙΑ ΤΟ ΔΙΑΔΙΚΤΥΟ

Αντώνης Μπικάκης

Μεταπτυχιακή Εργασία

Ηράκλειο, Σεπτέμβριος 2004

Πανεπιστήμιο Κρήτης
Σχολή Θετικών Επιστημών
Τμήμα Επιστήμης Υπολογιστών

Ανάπτυξη Συστήματος Αναιρέσιμων Κανόνων για το Διαδίκτυο

Εργασία που υποβλήθηκε από τον
Αντώνη Μπικάκη
ως μερική εκπλήρωση των απαιτήσεων για την απόκτηση
ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΕΙΔΙΚΕΥΣΗΣ

Συγγραφέας:

Αντώνης Μπικάκης
Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

Ηράκλειο, Σεπτέμβριος 2004

A System for Nonmonotonic Rules on the Web

Bikakis Antonis

Master of Science Thesis

Computer Science Department, University of Crete

Abstract

The Semantic Web is an extension of the current Web, in which information is given well-defined meaning, better enabling computers and people to work in cooperation. For the Semantic Web to function, computers must have access to structured collections of information and sets of inference rules that they can use to conduct automated reasoning. The development of the Semantic Web 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. The next step will be the logic and proof layer and *rule systems* appear to lie in the mainstream of such activities.

Until now, most studies on this domain have focused on the development of monotonic rule systems. A logic is *monotonic* if the truth of a proposition does not change when new information (axioms) are added to the system - the set of conclusions can only monotonically grow. In contrast, a logic is *non-monotonic* if the truth of a proposition may change when new information (axioms) is added - the set of conclusions may either grow or shrink. The study of nonmonotonic rule systems seem also to be promising for the development of the Web, as nonmonotonic reasoning is closer to commonsense reasoning, compared to monotonic logics.

In the current work, we are developing a nonmonotonic rule system, which is based on *defeasible logic*. The system can reason both with monotonic (*strict*) and nonmonotonic (*defeasible*) rules, and supports preferences between rules. It can treat facts in RDF form and ontologies in RDFS, and its user interface is compatible with

RuleML, the main standardization effort for rules on the Semantic Web. 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.

Ανάπτυξη Συστήματος Αναιρέσιμων Κανόνων για το Διαδίκτυο

Αντώνης Μπικάκης

Μεταπτυχιακή Εργασία

Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης

Περίληψη

Ο Σημασιολογικός Ιστός είναι μια επέκταση του σημερινού Ιστού, όπου η πληροφορία έχει καλά καθορισμένο νόημα, καθιστώντας τη συνεργασία μεταξύ υπολογιστών πιο εφικτή. Για τη λειτουργία του Σημασιολογικού Ιστού, οι υπολογιστές πρέπει να έχουν πρόσβαση σε δομημένες συλλογές πληροφοριών και σε σύνολα συμπερασματικών κανόνων που μπορούν να χρησιμοποιήσουν, ώστε να διεξάγουν αυτοματοποιημένη συλλογιστική. Η ανάπτυξη του Σημασιολογικού Ιστού βασίζεται σε πρότυπα γλωσσών δομημένων σε επίπεδα. Προς το παρόν, το υψηλότερο επίπεδο είναι αυτό των οντολογιών, στη μορφή γλωσσών που βασίζονται στην περιγραφική λογική. Το επόμενο βήμα θα είναι τα επίπεδα της συλλογιστικής και της τεκμηρίωσης, και τα συστήματα κανόνων καταλαμβάνουν κεντρική θέση σε αυτές τις δράσεις.

Μέχρι τώρα, οι περισσότερες μελέτες στον τομέα αυτό έχουν επικεντρωθεί στην ανάπτυξη μονοτονικών συστημάτων. Μία λογική χαρακτηρίζεται μονοτονική, αν η αλήθεια μίας πρότασης δεν αλλάζει όταν νέες πληροφορίες εισάγονται στο σύστημα – το σύνολο των συμπερασμάτων μπορεί μόνο να μεγαλώνει μονοτονικά. Μία λογική χαρακτηρίζεται μη μονοτονική αν η αλήθεια σε σχέση με μία πρόταση μπορεί να αλλάξει όταν προστίθενται νέες πληροφορίες – σε αυτήν την περίπτωση το σύνολο των συμπερασμάτων μπορεί να μεγαλώνει ή να συρρικνώνεται. Η μελέτη μη μονοτονικών συστημάτων κανόνων στο πλαίσιο της ανάπτυξης του Σημασιολογικού Ιστού φαίνεται να είναι εξίσου ενδιαφέρουσα, καθώς η μη μονοτονική συλλογιστική βρίσκεται πιο κοντά στη συλλογιστική της κοινής λογικής.

Στην παρούσα μελέτη, αναπτύσσουμε ένα μη μονοτονικό σύστημα κανόνων, που βασίζεται στην Αναιρέσιμη Λογική. Το σύστημα μπορεί να «συλλογίζεται» με βάση μονοτονικούς και αναιρέσιμος κανόνες, και υποστηρίζει προτεραιότητες μεταξύ των κανόνων. Μπορεί να χειρίζεται γεγονότα σε μορφή RDF και οντολογίες σε RDFS, και η διεπαφή του με το χρήστη είναι συμβατή με την RuleML, την κύρια προσπάθεια τυποποίησης κανόνων για τον Σημασιολογικό Ιστό. Στον πυρήνα του συστήματος συντελείται μετάφραση αναιρέσιμων θεωριών σε Prolog, με βάση μία μεθοδολογία που έχουμε αναπτύξει για την αναπαράσταση αναιρέσιμων σε λογικούς κανόνες. Ωστόσο, η υλοποίηση είναι δηλωτική, καθώς μεταφράζει τον τελεστή άρνησης χρησιμοποιώντας την Well-Founded σημασιολογία.

Ευχαριστίες

Αισθάνομαι την ανάγκη να ευχαριστήσω τον επόπτη καθηγητή μου, κύριο Γρηγόρη Αντωνίου, για την πολύτιμη καθοδήγησή του και την ουσιαστική συμβολή του στην ολοκλήρωση της εργασίας, αλλά και για την αποδοτική και ευχάριστη συνεργασία που είχαμε.

Θα ήθελα να ευχαριστήσω τον καθηγητή κύριο Βασίλη Χριστοφίδη, για τις πολύτιμες συμβουλές και παρατηρήσεις του, και για την συμμετοχή του στην εισηγητική επιτροπή της εργασίας, και την κυρία Αναστασία Αναλυτή για την συμμετοχή της στην εισηγητική επιτροπή.

Ευχαριστώ θερμά τους γονείς μου, Παντελή και Καλλιόπη, για την υλική και ψυχική στήριξη που μου παρείχαν καθ' όλη την διάρκεια της ακαδημαϊκής μου πορείας, και τον αδελφό μου Βαγγέλη για την ηθική υποστήριξη και τις πολύτιμες συμβουλές του.

Οφείλω να ευχαριστήσω, τέλος, τους φίλους μου και τους συμφοιτητές μου, για την ψυχολογική υποστήριξη, την συντροφία τους, και τις αμέτρητες στιγμές χαράς που μου προσέφεραν, ακόμα και στις πιο δύσκολες περιόδους της πορείας μου προς την ολολήρωση της εργασίας.

Table of Contexts

List of Figures

List of Tables

1 Introduction

1.1 Motivation and Contribution of the Study

The Semantic Web is an extension of the current Web, in which information is given well-defined meaning, better enabling computers and people to work in cooperation. For the Semantic Web to function, computers must have access to structured collections of information and sets of inference rules that they can use to conduct automated reasoning. The development of the Semantic Web [13] 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.

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.

So far, most studies have focused on the employment of monotonic logics in the layered development of the Semantic Web. A logic is *monotonic* if the truth of a proposition does not change when new information (axioms) are added to the system - the set of conclusions can only monotonically grow. In contrast, a logic is *non-monotonic* if the truth of a proposition may change when new information (axioms) is added - the set of conclusions may either grow or shrink. The study of nonmonotonic rule systems seem also to be promising for the development of the Web, as nonmonotonic reasoning is closer to commonsense reasoning, compared to monotonic logics.

Several nonmonotonic logics have been proposed and studied during the last decades: *default logic* [85], *autoepistemic logic* [71], *circumscription* [69] etc. 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 [5][73]and Courteous Logic Programs [42]. 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.

This thesis addresses the issues involved in 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 [88], 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. This translation enables reasoning on defeasible theories, using efficient logic programming systems, and exploits the well-founded semantics of logic programs.
- 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).
- It can reason both with monotonic and nonmonotonic rules. It can also treat facts in RDF form and ontologies in RDFS.

1.2 Thesis Organization

This report is organized as follows:

Chapter 2 presents the background in the domain of nonmonotonic reasoning: some classical nonmonotonic logics, the logic program semantics, argumentation systems, and systems with preferences.

Chapter 3 describes the role of the rule systems in the development of the Semantic Web. Firstly, we present the layers of the Semantic Web Tower that have been so far implemented, and then we reason why rule systems, and especially the nonmonotonic ones, are expected to be part of the layered development of the Semantic Web. Finally, we report on some web rule languages, that have so far been proposed in this direction.

In Chapter 4, we present the basic features, as well as some variants of Defeasible Logic. Specifically, we describe the ambiguity blocking variant and the ambiguity propagation variant, as well as the use of conflicting literals. In this Chapter, we also present a proof theory for this logic.

Chapter 5 is about the translation of defeasible theories into logic programs. We report on the rationale and some properties of the translation, in the case of each variant of defeasible logic.

Chapter 6 reports on the implementation architecture of our system. Firstly, we give an overview of how the system works, and then we describe in detail the functionality of each of the parts of the system.

In Chapter 7, we present a concrete example, showing in practice the abilities and the functionality of our system.

Chapter 8, reports on the results of the performance evaluation. Firstly, we describe the test theories that we used to conduct the evaluation, and then we compare the performance of our system, with the performance of other similar systems, based on the results of the experiments.

Finally, in Chapter 9 we present our conclusions and the plans of our future work.

2 Nonmonotonic Reasoning

Nonmonotonic reasoning is a subfield of Artificial Intelligence trying to find more realistic formal models of reasoning than classical logic. In common sense reasoning one often draws conclusions that have to be withdrawn when further information is obtained. The set of conclusions thus does not grow monotonically with the given information. It is this the phenomenon that nonmonotonic reasoning methods try to formalize.

In a monotonic logic system, given a collection of facts D that entail some sentence *s* (*s* is a logical conclusion of *D*), for any collection of facts *D'* such that $D \subseteq D'$, *D'* also entails *s*. In other words: *s* is also a logical conclusion of any superset of *D*.

In an nonmonotonic system, the addition of new facts can reduce the set of logical conclusions. So, if *s* is a logical conclusion of *D*, it is not necessarily a conclusion of any superset of *D.* Two of the basic characteristics of nonmonotonic systems are: *adaptability* (ability to deal with a changing environment), and the ability to reason under conditions of *uncertainty*. In other words, such systems are capable of adding and retracting beliefs as new sets of information is available, and reasoning with an incomplete set of facts.

McCarthy [68] was perhaps the first individual to discuss the need for the automation of commonsense reasoning, before any theory existed on the subject. Initial formalizations were suggested by McCarthy and Hayes [70], who discussed philosophical problems from the standpoint of AI and introduced the frame problem, and by Sandewall [89] who attempted to find a solution to the frame problem. The frame problem deals with how one specifies that when an action that is restricted to a set of objects takes place, the action has no effect upon many other objects in the world.

The *Prolog* language developed by Colmerauer and his students [27] and the *PLANNER* language developed by Hewitt [45] were the first languages to have a nonmonotonic component. The *not* operator in *Prolog*, and the *THNOT* capability in *PLANNER* provided default rules for answering questions about data where the facts did not appear explicitly in the program.

Reiter [83] set forth the rule of negation called the *closed world assumption, (CWA).* The *CWA* states that in Horn logic theories if we cannot approve an atom *p*, then we can assume *not p*. Clark [26] related negation to the *only if* counter-part of *if* statements in a logic program. The *if-and-only-if (iff)* statements form a theory in which negated atoms can be proven using a full theorem prover. The importance of Clark's observation is that he showed that for ground atoms and hierarchical programs, an inference system called *SLDNF* resolution, operating on the *if* statements of logic programs was sufficient to find the ground negated atoms in the *iff* theory that can be assumed *true*.

McCarthy first introduced his theory of *circumscription* in 1977 [69], and Doyle developed his *truth maintenance system* in 1979 [33]. Reiter gave preliminary material on default reasoning in 1978 [84]. The initial theories of nonmonotonic logic were presented in the *Artificial Intelligence Journal* in 1980.

Nonmonotonic logic has a rigorous mathematical basis. Although grounded in classical logic, it is a new discipline that extends classical logic and has become a mature part of logic. The main three classical approaches to nonmonotonic reasoning are Default Logic, Autoepistemic Logic and Circumscription. Recently research has also focused on an abstract study of nonmonotonicity, rather than the study of single approaches [53][65].

2.1 Classical Approaches

In this section we describe the main three classical approaches of nonmonotonic reasoning: default logic, autoepistemic logic and circumscription.

2.1.1 Default Logic

Reiter's *default logic* [85] is probably the most preeminent consistency-based nonmonotonic logic and has been used to formalize a number of different reasoning tasks, for instance, diagnosis from first principles [87] or inheritance [35].

Default Logic assumes that knowledge is represented in terms of a default theory. A default theory is a pair *(D,W). W* is a set of first order formulas representing the facts which are known to be true with certainty. *D* is a set of defaults of the form

$$A : B1,\ldots,Bn \Big/ C$$

where A, B_i, C are classical closed formulas. The default has the intuitive reading: if A is provable and, for all i ($1 \leq i \leq n$), $\neg B_i$ is not provable, then derive C. A is called the *prerequisite*, B_i a *consistency condition* or *justification*, and C the *consequent* of the default. We can make this clear by giving an example form the legal domain: According to the German law, a foreigner is usually expelled if he has committed a crime. One of the exceptions to this rule concerns political refugees. This information is expressed by the default

$$\frac{criminal(X) \land foreigner(X) : expel(X)}{expel(X)}$$

in combination with the rule

$$politicalRefugee(X) \rightarrow \neg expel(X).$$

Default rules act as mappings from some incomplete theory to a more complete *extension* of the theory. An extension is a maximal set of conclusions that can be drawn from the default theory. Extensions are defined by a fixed point construction. For any set of first-order sentences S, define $\Gamma(S)$ to be the smallest set satisfying the following three properties:

1. $W \subset \Gamma(S)$.
2. $\Gamma(S)$ is closed under first-order logical consequence.
3. If $\alpha : \beta / \gamma$ is a default rule of D and $\alpha \in \Gamma(S)$ and $\neg\beta \notin S$ then $\gamma \in \Gamma(S)$.

Then E is defined to be an *extension* of the default theory (D,W) iff $\Gamma(E) = E$, that is, E is a fixed point of the operator Γ. A theory consisting of general default rules does not always have an extension. A subclass consisting of default rules called *normal defaults*, and of the form, $\alpha{:}\beta \,/\, \beta$ always has en extension. Reiter develops a complete proof theory for normal defaults and shows how it interfaces with a top-down resolution theorem prover.

Reiter and Criscuolo [86] show that default rules may be normal when viewed in isolation, however they can interact in ways that lead to derivations of anomalous default assumptions. Non-normal defaults are required to deal with default interactions. In general, non-normal defaults are more difficult to implement and reason with.

Gelfond, Lifschitz, Przymusinska and Truszczynski [37] generalize Reiter's default logic to handle disjunctive information. The generalization was motivated by a difficulty encountered in attempts to use defaults in the presence of disjunctive information. The difficulty has to do with the difference between a default theory with two extensions – one containing a sentence a, the other a sentence β – and the theory with a single extension, containing the disjunction $a \vee \beta$. A disjunctive default is an expression of the form

$$\frac{a : \beta_1, \ldots, \beta_m}{\gamma_1 \mid \ldots \mid \gamma_n}$$

where α, β_ι, γ_ι are quantifier free formulas. A *disjunctive default theory* (*ddt*) is a set of disjunctive defaults. Gelfond et al. [37] show that one cannot simulate a *ddt* with a standard default theory.

Defaults can be used naturally to model the *Closed World Assumption* which is used in database theory, algebraic specification, and logic programming. According to this assumption, a ground fact is taken to be false in a problem domain if it does not follow from the axioms (in the form of relational facts, equations, rules etc.) describing the domain. The *CWA* has the simple default representation

$$\frac{true : \neg\phi}{\neg\phi}$$

for each ground atom φ. The explanation of the default is: if it is consistent to assume $\neg\phi$ (which is equivalent to not having a proof for φ then conclude $\neg\phi$).

In [25], it is reported that showing the existence of an extension has been proved to be $\sum_2^p$ - complete for semi-normal default theories. Skeptical default reasoning in the prerequisite-free normal case has been shown to be $\prod_2^p$ - complete.

2.1.2 Autoepistemic Logic

Moore's [71] *Autoepistemic Logic* (*AEL*) is the most widely studied logic of a class called *modal nonmonotonic logics.* These logics use a modal operator to express explicitly that a certain formula is consistent or believed. Moore introduces the modal operator *L* into the logical language: in other words, if *p* is a (closed) formula then also *Lp* is. *Lp* stands for "*p* is believed". The idea is to model an ideal introspective agent reasoning about his beliefs. Introspective here means that the agent knows completely what he knows and, in particular, what he does not know. This means that

- if a formula p belongs to the set of beliefs B of the agent, then also Lp has to belong to B, and
- if p does not belong to B, then $\neg Lp$ must be in B.

Nonmonotonicity in this framework arises whenever the derivation of a formula depends on disbelief in a formula. Consider the following representation of the famous bird rule:

$$Bird(Tweety) \wedge \neg L \neg Flies(Tweety) \supset Flies(Tweety).$$

Assume *Bird(Tweety)* is all the agent knows about Tweety. Then, since the negation of this formula $\neg Flies(Tweety)$, is not believed, $\neg L \neg Flies(Tweety)$ must be contained in the agent's beliefs. Applying modus ponens to the bird rule, we thus can derive *Flies(Tweety).* On the other hand, if the agent obtains new information about Tweety is a non-flying bird, then $\neg L \neg Flies(Tweety)$ will not be contained in his set of beliefs, and the former conclusion is withdrawn.

As in *Default Logic*, conflicting *AEL* rules can lead to alternative stable sets of beliefs a reasoner may adopt. Moore called these sets expansions. Konolige [50] later introduced a somewhat different fixed point equation, called the fixed points extensions, and showed that Moore's expansions and his extensions are equivalent. According to Konolige:

Let A be a set of AEL formulas. T is an extension of A if

$$T = \{p \mid A \cup Bel(T) \cup Disbel(T) \vdash p\},$$

where $Bel(T) = \{Lq \mid q \in T\}$*, and* $Disbel(T) = \{\neg Lq \mid q \notin T\}$.

Although the intuitions underlying *DL* and *AEL* seem very different at first sight, these two logics are actually much closer than one might expect. In [50], Konolige used the following translation from *DL* defaults to *autoepistemic* formulas:

$$A{:}B_1,\dots,B_n / C \Rightarrow LA \wedge \neg L \neg B_1 \wedge \dots \wedge \neg L \neg B_n \supset C.$$

Since on the other hand, every *AEL* formula can be transformed into an *AEL* implication of this kind shown above, the translation also works from *AEL* to *DL*.

Konolige showed that extensions to a default theory *(D,W)* correspond exactly to the objective part, that is the part not containing the modal operator *L*, of some of the extensions of the *AEL* translation of *(D,W)*. He called these extensions strongly grounded extensions. The *AEL* extensions that do not correspond to *DL* extensions contain beliefs that depend on themselves. For instance, the *AEL* theory $\{Lp \supset p\}$ has two extensions, one containing *Lp* and *p*, and the other not containing *p*, and therefore containing $\neg Lp$.

More recently other translations from *DL* to *AEL* have been investigated [66], [97]. In the latter paper, the following translation is proposed.

$$A{:}B_1,\ldots,B_n / C \Rightarrow LA \wedge L\neg L\neg B_1 \wedge \ldots \wedge L\neg L\neg B_n \supset C.$$

It is shown that under this translation *DL* can be embedded into a whole range of nonmonotonic modal logics. However, this translation only works in one direction, from *DL* to *AEL*, since not every *AEL* formula can be equivalently transformed to the above form.

One of the main limitations of autoepistemic logic is that quantification into the scope of the *L* operator is not allowed. [51] extends the logic in this direction. The same author developed the *Hierarchic Autoepistemic Logic* [52] to deal with some deficiencies of *AEL*: to allow priorities among formulae, and to increase computational efficiency. The main idea is to consider a collection of subtheories linked together in a hierarchy, and to use a set of *L* operators with restricted application scope. Though new representational problems arise, [8] shows an interesting application.

In [25], it is reported that the results about the computational complexity of autoepistemic logic are exactly the same already obtained for default logic; the problem of deciding whether a set of premises *A* has a stable expansion is $\sum_2^p$ -complete, while skeptical autoepistemic reasoning is $\prod_2^p$ - complete.

2.1.3 Circumscription

Circumscription has generated a great deal of interest in the nonmonotonic reasoning community. McCarthy's 1980 paper on circumscription is a formalization of the work he described first in [69]. *Circumscription* deals with the minimization of predicates subject to restrictions expressed by predicate formulas. In circumscription, theories are written in classical first-order logic, however the entailment relation is not classical. A formula F is entailed by a circumscriptive theory T if it is true in all minimal models of T. A model M, of a theory, T, is called minimal with respect to some ordering, $<$, of models if there is no model, N, of T such that $N < M$. A circumscription policy is used to determine a particular partial ordering, $<$, used to circumscribe the theory. In the basic case a single predicate, P, is chosen to be circumscribed. Given two models, M_1 and M_2, which differ only in the interpretation of P, we say that $M_1 \leq M_2$ if the extent of P in M_1 is a subset of its extent in M_2. We write $circ(T; P) \models F$ to indicate that F is entailed form T by circumscription with the policy of minimizing P. Here $circ(T; P)$ can be viewed as a second-order formula expressing the above definition.

Below we apply this idea to a version of the flying birds example. We can express that Tweety is a bird and birds normally fly by first-order sentences

$$T = \{bird(tweety)\ ,\ \ \forall X\ (bird(X) \wedge \neg\, ab(X)\ \supset\ fly(X))\}$$

where $ab(X)$ means that X is abnormal with respect to flying. Obviously, classical first-order logic does not allow us to reach the desired common-sense conclusion that Tweety can fly. If, however, we use circumscription and circumscribe ab, then all minimal models under this policy contain $fly(tweety)$ and hence

$$circ(T;\ ab) \models fly(tweety).$$

The basic form of circumscription is often too restrictive so many other circumscriptive policies have been formalized. One common policy is to specify certain predicates which are allowed to vary. In this case, models are comparable if they differ in the extent of the varied predicates as well as the circumscribed predicate. As before, the ordering of comparable models is based solely on the extent of the circumscribed predicate in models. Suppose we add to the above example the fact that penguins are birds which are abnormal with respect to flying

$$(penguin(X)\ \supset\ bird(X) \text{ and } penguin(X)\ \supset\ ab(X)).$$

Since a model, M_0, in which Tweety is a penguin, is minimal with respect to ordering defined by the first policy we no longer have $circ(T;\ ab) \models fly(tweety)$. If we modify the policy so that *penguin* can vary, the model M_0 will not be minimal with respect to the new ordering. It is easy to check that, under this policy, T concludes *fly(tweety).* Selection of the right circumscriptive policy lies at the heart of representing knowledge in circumscriptive theories. Even though computing consequences of circumscribed theories is generally intractable (even in propositional case), for some theories there are reasonably efficient algorithms based on reducing the circumscribed theory to a logic program or a set of first-order formulae.

2.2 Extended Logic Program Semantics

A (propositional) extended logic program consists of rules of the form

$$c \leftarrow a_1,\ldots,a_n,\ not\ b_1,\ldots,\ not\ b_m$$

where the a_i, b_j and c are propositional literals, i.e., either propositional atoms or such atoms preceded by the classical negation sign. The symbol *not* denoted negation by failure (weak negation), $\neg$ denotes classical (strong) negation. A rule schema is used to represent a set of propositional rules, namely the set of all ground instances of the schema. Extended logic programs are very useful for knowledge representation purposes. In [11] one can find a number of illustrative examples. Two major semantics for extended logic programs have been defined: answer set semantics [36], an extension of stable model semantics, and a version of well-founded semantics [82].

2.2.1 Answer set semantics [36]

We say a rule $r \in P$ of the form above is defeated by a literal l if $l = b_i$ for some $i \in \{1,\ldots,m\}$. We say r is defeated by a set of literals X if X contains a literal that defeats r. Furthermore, we call the rule obtained by deleting weakly negated preconditions form r the monotonic counterpart of r and denote it with *Mon(r).* We also apply *Mon* to sets of rules with the obvious meaning.

Definition 1 *Let P be a logic program, X a set of literals. The X-reduct of P, denoted P^X, is the program obtained form P by*

- *deleting each rule defeated by X, and*
- *replacing each remaining rule r with its monotonic counterpart Mon(r).*

Definition 2 *Let R be a set of rules without negation as failure. Cn(r) denotes the smallest set of literals that is closed under R, and logically closed, i.e., either consistent or equal to the set of all literals.*

Definition 3 *Let P be a logic program, X a set of literals. Define an operator γ_P as follows:*

$$\gamma_P(X) = Cn(P^X)$$

X is an answer set of P iff $X = \gamma_P(X)$.

A literal l is a consequence of a program P under answer set semantics, denoted $l \in Ans(P)$, if l is contained in all answer sets of P.

For example, consider the following program

$$\begin{array}{rcl} flies(X) & \leftarrow & bird(X),\ not\ \neg flies(X) \\ \neg flies(X) & \leftarrow & penguin(X),\ not\ flies(X) \\ bird(X) & \leftarrow & penguin(X) \\ penguin(tweety) & \leftarrow & \end{array}$$

The program has two answer sets, namely:

$$I_1 = \{ flies(tweety), bird(tweety), penguin(tweety)\}$$

$$I_2 = \{ \neg flies(tweety), bird(tweety), penguin(tweety)\}$$

An extended program is called *contradictory* with respect to the answer-set semantics if it has no consistent answer sets. For example, the program containing the two facts a and $\neg a$ has a single answer-set $\{ a, \neg a\}$ which is inconsistent. So this program is contradictory.

2.2.2 Well-founded Semantics [82]

Like answer set semantics the well-founded semantics for extended logic programs is based on the operator γ_P. However, the operator is used in a totally different way. Since γ_P is anti-monotone the function $\Gamma_P = (\gamma_P)^2$ is monotone. According to the famous Knaster-Tarski theorem [92] every monotone operator has a least fixpoint. The set of well-founded conclusions of a program P, denoted $WFS(P)$, is defined to be this least fixpoint of Γ_P. The fixpoint can be approached from below by iterating Γ_P on the empty set. In case P is finite this iteration is guaranteed to actually reach the fixpoint. The intuition behind this use of the operator is as follows: whenever γ_P is applied to a set of literals X known to be true it produces the set of all literals that are still potentially derivable. Applying γ_P again to such a set of potentially derivable literals it produces a set of literals known to be true, often larger than the original set X. Starting with the empty set and iterating until the fixpoint is reached thus produces a set of true literals. It can be shown that every well-founded conclusion is a skeptical conclusion under the answer set semantics. Well-founded semantics can thus be viewed as an approximation of answer set semantics.

2.3 Argumentation Systems

Argumentation systems are yet another way to formalise nonmonotonic reasoning, viz. as the construction and comparison of arguments for and against certain conclusions. In these systems the basic notion is not that of a defeasible conditional but that of a defeasible argument. The idea is that the construction of arguments is monotonic, i.e., an argument stays an argument if more premises are added. Nonmonotonicity, or defeasibility, is not explained in terms of the interpretation of a defeasible conditional, but in terms of the interactions between conflicting arguments: in argumentation systems nonmonotonicity arises from the fact that new premises may give rise to stronger counter-arguments, which defeat the original argument. So in case of Tweety we may construct one argument that Tweety flies because it is a bird, and another argument that Tweety does not fly because it is a penguin, and then we may prefer the latter argument because it is about a specific class of birds, and is therefore an exception to the general rule.

Argumentation systems can be applied to any form of reasoning with contradictory information, whether the contradictions have to do with rules and exceptions or not. For instance, the contradictions may arise from reasoning with several sources of information, or they may be caused by disagreement about beliefs or about moral, ethical or political claims. Moreover, it is important that several argumentation systems allow the construction and attack of arguments that are traditionally called ‘ampliative’, such as inductive, analogical and abductive arguments; these reasoning forms fall outside the scope of most other nonmonotonic logics.

Most argumentation systems have been developed in artificial intelligence research on nonmonotonic reasoning, although Pollock’s work [76], which was the first logical formalisation of defeasible argumentation, was initially applied to the philosophy of knowledge and justification (epistemology). The first artificial intelligence paper on argumentation systems was [60]. Argumentation systems have been applied to domains such as legal reasoning, medical reasoning and negotiation. Below, we present an abstract approach to defeasible argumentation, developed in several articles by Bondarenko, Dung, Toni and Kowalski. We also give a brief description of some other interesting approaches.

2.3.1 The Abstract Approach of Bondarenko, Dung, Kowalski and Toni

Historically, this work came after the development by others of a number of argumentation systems (to be discussed below). The major innovation of the BDKT approach is that it provides a framework and vocabulary for investigating the general features of these other systems, and also of nonmonotonic logics that are not argument-based.

The latest and most comprehensive account of the BDKT approach is Bondarenko *et al.* [15]. In this account, the basic notion is that of a set of "assumptions". In their approach the premises come in two kinds: 'ordinary' premises, comprising a *theory*, and *assumptions*, which are formulas (of whatever form) that are designated (on whatever ground) as having default status. Bondarenko *et al.* [15] regard nonmonotonic reasoning as adding sets of assumptions to theories formulated in an underlying monotonic logic, provided that the contrary of the assumptions cannot be shown. What in their view makes the theory argumentation-theoretic is that this provision is formalised in terms of sets of assumptions attacking each other. In other words, according to Bondarenko *et al.* [15] an argument is a set of assumptions. This approach has especially proven successful in capturing existing nonmonotonic logics.

Another version of the BDKT approach, presented by Dung [34], completely abstracts from both the internal structure of an argument and the origin of the set of arguments; all that is assumed is the existence of a set of arguments, ordered by a binary relation of 'defeat'. Dung then defines various notions of so-called argument extensions, which are intended to capture various types of defeasible consequence. These notions are declarative, just declaring sets of arguments as having a certain status. Finally, Dung shows that many existing nonmonotonic logics can be reformulated as instances of the abstract framework.

2.3.2 Other Approaches

Pollock. Another interesting approach is Pollock's argumentation system [76]. In this system, the underlying logical language is standard first-order logic, but the notion of an argument has some non-standard features. What still conforms to accounts of deductive logic is that arguments are sequences of propositions linked by inference rules (or better, by instantiated inference schemes). However, Pollock's formalism begins to deviate when we look at the kinds of inference schemes that can be used to build arguments.

Inheritance systems. A forerunner of argumentation systems is work on so-called inheritance systems, especially of Horty *et al.* [48], which we shall briefly discuss. Inheritance systems determine whether an object of a certain kind has a certain property. Their language is very restricted. The network is a directed acyclic graph. Its initial nodes represent individuals and its other nodes stand for classes of individuals. There are two kinds of links $\rightarrow$ and $\nrightarrow$, depending on whether an individual does or does not belong to a certain class, or a class is or is not member of a certain class. Links from an individual to a class express class membership, and links between two classes express class inclusion. A path through the graph is an *inheritance path* iff its only negative link is the last one. Thus the following are examples of inheritance paths:

P_1: Tweety $\rightarrow$ Penguin $\rightarrow$ Canfly

P_2 : Tweety $\rightarrow$ Penguin $\nrightarrow$ Canfly

Another basic notion is that of an assertion, which is of the form

$$x \rightarrow y \qquad \text{or} \qquad x \nrightarrow y$$

where x is an individual and y is a class. Such an assertion is *enabled* by an inheritance path if the path starts with *x* and ends with the same link to *y* as the assertion. Above, an assertion enabled by P_1 is Tweety $\rightarrow$ Canfly, and an assertion enabled by P_2 is Tweety $\nrightarrow$ Canfly. Two paths can be conflicting. They are compared on specifity, which is read off from the syntactic structure of the net, resulting in relations of *neutralisation* and *preemption* between paths. Although Horty *et al.*

present their system as a special-purpose formalism, it clearly has all the elements of an argumentation system. An inheritance path corresponds to an argument, and an assertion enabled by a path to a conclusion of an argument. Their notion of conflicting paths corresponds to rebutting attack. Furthermore, neutralisation and preemption correspond to defeat, while a permitted path is the same as a justified argument.

Lin & Shoham. Before the BDKT approach, an earlier attempt to provide a unifying framework for nonmonotonic logics was made by Lin & Shoham [59]. They show how any logic, whether monotonic or not, can be reformulated as a system for constructing arguments. However, in contrast with the other theories in this section, they are not concerned with comparing incompatible arguments, and so their framework cannot be used as a theory of defeat among arguments. The basic elements of Lin & Shoham's abstract framework are an unspecified logical language, only assumed to contain a negation symbol, and an also unspecified set of inference rules defined over the assumed language. Arguments can be constructed by chaining inference rules into trees. Inference rules are either monotonic or nonmonotonic.

Vreeswijk. Like the BDKT approach and Lin & Shoham [59], Vreeswijk [99], [100] also aims to provide an abstract framework for defeasible argumentation. His framework builds on the one of Lin & Shoham, but contains the main elements that are missing in their system, namely, notions of conflict and defeat between arguments. As Lin&Shoham, Vreeswijk also assumes an unspecified logical language *L*, only assumed to contain the symbol $\perp$, denoting 'falsum' or 'contradiction,' and an unspecified set of monotonic and nonmonotonic inference rules (which Vreeswijk calls 'strict' and 'defeasible'). This also makes his system an abstract framework rather than a particular system.

Prakken & Sartor. Inspired by legal reasoning, Prakken & Sartor [80], [81] have developed an argumentation system that combines the language (but not the rest) of default logic with the grounded semantics of the BDKT approach. Actually, Prakken & Sartor originally used the language of extended logic programming, but Prakken [79] generalised the system to default logic's language. The main contributions to defeasible argumentation are a study of the relation between rebutting and assumption attack, and a formalisation of argumentation about the criteria for defeat.

2.4 Defeasible Logics

A development closely related to defeasible argumentation is so-called ‘defeasible logic’, initiated by Donald Nute [73] . In both fields the notion of defeat is central. However, while in defeasible argumentation defeat is among arguments, in defeasible logic it happens between rules, resulting in lower computational complexity. Defeasible logics capture key ideas of inheritance networks in logical rule systems.

Nute’s systems are based on the idea that defaults are not propositions but inference licenses. Thus Nute’s defeasible rules are, like Reiter’s defaults, one directional. In general, Nute’s defeasible rules correspond to normal default rules. However, unlike Reiter’s defaults they are twoplace; assumption attacks are dealt with by an explicit category of defeater rules.

As for the underlying logical language, since Nute’s aim is to develop a logic that is efficiently implementable, he keeps the language as simple as possible. In general, a defeasible theory (a knowledge base in defeasible logic) consists of five different kinds of knowledge: facts, strict rules, defeasible rules, defeaters, and a superiority relation.

Facts are indisputable statements, for example, “Tweety is an emu”. Written formally, this would be expressed as

$$emu(tweety).$$

Strict rules are rules in the classical sense: whenever the premises are indisputable (e.g., facts) then so is the conclusion. An example of a strict rule is “Emus are birds”. Written formally:

$$emu(X) \rightarrow bird(X).$$

Defeasible rules are rules that can be defeated by contrary evidence. An example of such a rule is “Birds typically fly”; written formally:

$$bird(X) \Rightarrow flies(X).$$

The idea is that if we know that something is a bird, then we may conclude that it flies, *unless there is other, not inferior, evidence suggesting that it may not fly.*

Defeaters are rules that cannot be used to draw any conclusions. Their only use is to prevent some conclusions. In other words, they are used to defeat some defeasible rules by producing evidence to the contrary. An example is “If an animal is heavy then it might not be able to fly”. Formally:

$$heavy(X) \rightsquigarrow \neg flies(X).$$

The main point is that the information that an animal is heavy is not sufficient evidence to conclude that it does not fly. It is only evidence that the animal *may* not be able to fly. In other words, we do not wish to conclude $\neg flies(X)$ if $heavy(X)$; we simply want to prevent a conclusion $flies(X)$.

The *superiority relation* among rules is used to define priorities among rules, i.e., where one rule may override the conclusion of another rule. For example,

given the defeasible rules

$$r: bird(X) \Rightarrow flies(X)$$

$$s: brokenWing(X) \Rightarrow \neg flies(X)$$

which contradict one another, no conclusive decision can be made about whether a bird with broken wings can fly. But if we introduce a superiority relation > with $s > r$, with the intended meaning that s is strictly stronger than r, then we can indeed conclude that the bird cannot fly.

Notice that a cycle in the superiority relation is counterintuitive. In the above example, it makes no sense to have both $r > s$ and $s > r$. Consequently, we focus on cases where the superiority relation is acyclic.

Another point worth noting is that, in Defeasible Logic, priorities are *local* in the following sense: two rules are considered to be competing with one another only if they have complementary heads. Thus, since the superiority relation is used to resolve conflicts among competing rules, it is only used to compare rules with complementary heads; the information $r > s$ for rules r, s without complementary heads may be part of the superiority relation, but has no effect on the proof theory.

A more formal definition of Defeasible Logic and a proof theory are given in Chapter 4.

2.5 Systems with Preferences

The notion of *preference* is pervasive in commonsense reasoning, in part because preferences constitute a very natural and effective way of resolving indeterminate situations. In decision making, for example, one may have various desiderata, not all of which can be simultaneously satisfied; in such a situation, preferences among desiderata may allow one to come to an appropriate compromise solution. In legal reasoning, laws may conflict. Conflicts may be resolved by principles such as ruling that newer laws will have priority over less recent ones, and laws of a higher authority will have priority over laws of a lower authority. For a conflict among these principles, one may further decide that the “authority” preference takes priority over the “recency” preference.

Preference has a decidedly nonmonotonic flavour. Or, more accurately, it may be considered as having a *fundamental* nonmonotonic aspect. Given a preference ordering, however constituted, and some basic or case-specific information, Ψ, one may come up with a set of desired outcomes. However, a superset of this case-specific information, $\Psi \cup \Phi$, may lead to a different set of desired outcomes. For example, imagine feeding information into an automated financial advisor: that one is a relatively cautious investor, that one has a long-term horizon, etc. Given these preferences, a set of recommended mutual funds may be suggested by the automated advisor. If the user subsequently states that they also prefer that their funds invest in environmentally and socially responsible companies, then a different set of suggestions may well result.

The source of information about preference in a knowledge modelling system may be varying:

- Preferences may be part of the domain being modelled. For example, the legal domain contains principles, regulations, rules with exceptions etc.[78].
- Preferences may arise dynamically during the deductive process. Consider for example, a dynamic preference between two rules r_1 and r_2, inferred by another rule of the form: *if A* then r_1 *is superior to* r_2 [18][19].
- Information about preference may be extracted form the knowledge base, e.g. based on specifity [77][94][96].

In logic-based AI, a standard approach to handling preferences is to take an existing system of nonmonotonic reasoning and, in one fashion or another, equip it with preferences. For example, preferences are added in such a manner to default logic [22],[32] autoepistemic logic [52], circumscription [58], and logic programming [103], [21]. However, although the notion of "preference" is intuitively straightforward, there is a surprising variety in how this notion is realised in various approaches. Thus, some approaches take a preference ordering as expressing a "desirability" that a property be adopted while in others the ordering expresses the order in which properties (or whatever) are to be considered. Some approaches conflate the notion of *inheritance of properties* with the general notion of preference. The outcome of course is that, depending on how the notion of preference is interpreted, different conclusions may be forthcoming. At the same time, while logical preference handling already constitutes an indispensable means for legal reasoning systems [79], it is also being used in other application areas such as intelligent agents and e-commerce [41], and the resolution of grammatical ambiguities [29].

In [31], Delgrande *et. al* present an overview and classification for approaches to dealing with preference. They classify the approaches based on a set of criteria. These criteria include:

- the host system, meaning the system that is extended to support preferences. In most approaches, default logic is selected, as it is considered to be the most popular reasoning formalism. More recently, systems use extended logic programs as the "underlying" system.
- meta-level vs. object-level preferences: whether the preference ordering is imposed "externally" on rules of the system, or it is imposed at the object-level. In the first case, the underlying host system is used more or less as a black box by an enveloping preference system. In contrast, an object-level approach to preference allows the use of preferences within the object theory. The advantage of the first approach is that it is much easier to realise. On the other hand the object-level approach is more flexible, and the preferences are formalized *within* a theory, instead of *about* a theory.

- static vs. dynamic preferences: whether the preferences are *static*, or fixed at the time the theory is specified, or *dynamic*, and so can be determined "on the fly".
- properties of the preference ordering: whether the superiority relation is a partial or a total order.
- the *specific kind* of objects that the superiority relation is applied upon: the rules themselves or the name of the rules.
- *prescriptive* vs. *descriptive* preferences: In the first case, the ordering of the preferences specifies the order in which the rules are applied. In the latter one, the preference order represents a ranking on desired outcomes: the desirable situation is one where the most preferred rules are applied.

Their classification includes interesting approaches, such as:

- the **Brewka & Eiter** approach [22]. This is a default logic – based, meta-level, semi-perspective approach, which uses static preferences on rules and strict partial order, with complexity equal to the complexity of default logic.
- the **Delgrande & Schaub** approach [32]. This approach is also based on default logic. It uses dynamic meta-level preferences with strict partial order, and its complexity is equal to the complexity of the host system.
- the **Konolige** approach [52]. It uses autoepistemic logic as its host system. It is a met-level descriptive approach, where the preferences are expressed as layered sets of formulas.
- the **Lifschitz** approach [58]. It is based on circumscription, and uses meta-level static preferences on special-purpose predicates.
- the **Zhang and Foo** approach [103]. Its host system is extended logic programs under answer sets. The strategy that it uses is modified answer sets. The preferences, which are meta-level and dynamic, are applied on rules. Its complexity is higher than that of the host system.
- the **Brewka & Eiter** approach [21]. This approach is also based on extended logic programs under answer sets. It uses meta-level, semi-prescriptive, static

preferences with partial order. Its complexity is equal to the complexity of the host system.

- the **Alferes & Pereira** approach [2]. Its host system is dynamic logic programs. It uses semi-perspective, meta-level preferences with strict partial order. The preferences are applied on rules, and its complexity is equal to that of its host system.
- the **defeasible logic** approach [73],[14],[4]. This approach is based on defeasible logic. It uses meta-level, perspective and static preferences in an arbitrary order. The preferences are applied on rules.

3 Rules for the Semantic Web

"The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation".

This is an informal definition for the Semantic Web, given by Berner-Lee in the May 2001 American article "The Semantic Web" [13]. The Semantic Web is an extension of the World Wide Web in which both data and its semantic definition can be processed by computer programs. The next generation of the Web will combine existing Web technologies with knowledge representation formalisms in order to provide an infrastructure allowing data to be processed, discovered and filtered more effectively on the Web. A set of new languages organized in a layered architecture will allow users and applications to write and share information in a machine-readable way, and will enable the development of a new generation of technologies and toolkits. This layered architecture of the Semantic Web is often referred to as the Semantic Web tower [13].

3.1 The Semantic Web Tower

The Semantic Web tower (Figure 3.1) is a work in progress, where the layers are developed in a bottom-up manner. The so far defined languages in the bottom-up order include: XML, RDF, RDF Schema and Web Ontology Language OWL. The next step in the development of the Semantic Web will be the logic and proof layer. In the next sections we will briefly describe the basic layers of the tower.

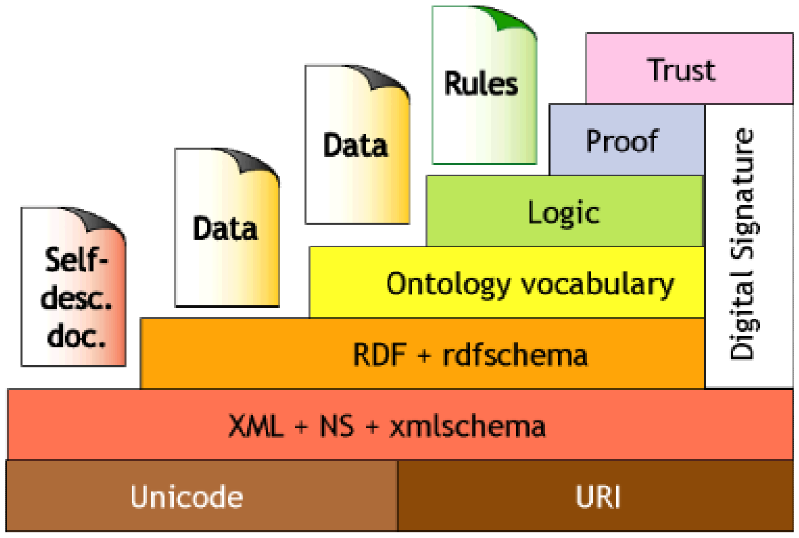


Figure 3.1: The Semantic Web Tower

3.1.1 The Representation Layer

The base language of the Semantic Web tower is XML [17], the dominant standard for data encoding and exchange on the web. Essentially the data represented in XML can be seen as labelled ordered trees. Such trees are encoded as XML documents with the parenthesis structure marked by tags. In the context of the Semantic Web XML will be used for encoding any kind of data, including the meta-data, describing the meaning of application data. Such meta-data will be described by the languages of the next layers of the Semantic Web tower.

Several mechanisms have been proposed for defining sets of XML documents. A standard one is the XML Schema language [93]. The elements of this language, called XML schemas, are XML documents. Thus, an XML schema is an XML document defining a (usually infinite) set of XML documents. This makes possible automatic validation of a given XML document d with respect to a given schema s, that is automatic check, whether or not d is in the set of documents defined by s.

The syntax of the languages of the next layers of the Semantic Web is also defined in XML. This means that the constructs of these languages are encoded as XML documents, and can be validated against the language definitions by standard validators. However, alternative syntaxes, better suitable for the human, can be provided and can be used as a starting point for defining the semantics of these languages.

The XML Namespaces [16] and Uniform Resource Identifiers [12] are important standards used in XML and therefore also in the upper layers of the Semantic Web, which are encoded in XML. They make it possible to create unique names for web resources. In the upper layers of the Semantic Web such names may be used as logical constants.

3.1.2 RDF and Ontology Languages

The idea of the Semantic Web is to describe the meaning of web data in a way suitable for automatic reasoning. This means that a descriptive data (meta-data) in machine readable form is to be stored on the web and used for reasoning. The simplest form of such description would assert relations between web resources. A more advanced description, called ontology, to be shared by various applications, would define concepts in the domain of these applications. Usually an ontology defines an hierarchy of classes of objects in the described domain and binary relations, called properties.

The Semantic Web tower introduces language layers for describing resources and for providing ontologies:

The Resource Description Framework (RDF) [54] makes it possible to assert binary relations between resources (identified by URI's), and between resources and literals, which are strings. Such assertions have the form of triples, called statements. The elements of a triple are called subject, predicate (or property), and object. Usually they are URI references; the object may also be a literal. A triple can be seen as a kind of an atomic formula with a binary predicate. However, the vocabulary of RDF does not distinguish predicate symbols from logical constants: the predicates of RDF sentences may also appear as subjects and objects. In addition, RDF allows reification of a statement which can then for example be used as the subject of another statement.

For describing hierarchies of concepts RDF is extended with some built-in properties interpreted in a special way. The extension is called RDF Schema [23]. Statements of RDF Schema (RDFS) make it possible to define hierarchies of classes, hierarchies of properties and to describe domains and ranges of the properties. RDFS allows defining simple ontologies without using advanced features of RDF, like reification.

The emerging Web Ontology Language OWL [75] builds-up on RDFS introducing more expressive description constructs. However, as explained in [74], defining an expressive ontology language as a semantic extension of RDFS may lead to paradoxes. The design of OWL takes this into account. OWL has three increasingly expressive sublanguages: OWL Lite, OWL DL and OWL Full. *OWL Lite* supports those users primarily needing a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. It should be simpler to provide tool support for OWL Lite than its more expressive relatives, and OWL Lite provides a quick migration path for thesauri and other taxonomies. The complexity of computing ontology entalment is also lower for OWL Lite, than OWL DL. *OWL DL* supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class). OWL DL is so named due to its correspondence with *description logics* [10]. *OWL Full* is meant for users who want very high expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary.

3.1.3 The Top Layers

The top three layers of the Semantic Web tower are: the logic layer, the proof layer and the trust layer. The *logic* layer is used to enhance the ontology language further, and to allow to write application-specific declarative knowledge. The *proof* layer involves the actual deductive process, as well as the representation of proofs in Web languages and proof validation. Finally *trust* will merge through the use of *digital signatures*, and other kinds of knowledge, based on recommendations by agents we trust, on rating and certification agencies and on consumer bodies. Being located at the top of the pyramid, trust is a high-level and crucial concept: The Web will only achieve its full potential when users have trust in its operations (security) and the quality of the information provided.

3.2 The Role of the Rules

Rules constitute the next, not yet developed language level over the ontology languages in the Semantic Web tower. The arguments supporting the need of rules in the Semantic Web include the following:

- Rules appear naturally in many applications, e.g. business rules, policy specifications, service descriptions, database queries and many others. It is desirable to have a web rule language for expressing them for web applications.
- Rules provide a high-level description, abstracting from implementation details; they are concise and simple to write. They are well-known, understood

by non-experts, and well integrated in the mainstream Information Technology.

- The ontology languages are designed to describe concepts of the application domains, but are not sufficiently expressive for describing some aspects of applications, expressible in rule languages, e.g. composition of relations, extensively used in database query languages.

The ongoing discussion on rules for the Semantic Web seems to indicate that a family of rule languages may be needed rather than one language, since different applications require different kind of rules. The effort to define such languages and to enable Web-based interoperability between various rule systems and applications has been undertaken by the RuleML Initiative [88], which is further discussed below.

In general, the role that the rule systems are expected to have in the development of the Semantic Web is twofold:

(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.

Possible interactions between description logics and monotonic rule systems were studied in [43]. Based on that work and on previous work on hybrid reasoning [55], 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.

3.2.1 The Role of Nonmonotonic Rule Systems

One of the issues that have recently attracted the concentration of the developers of the Semantic Web, is the nature of the rule systems that should be employed in the logic layer of the Semantic Web tower. Monotonic rule systems have already been studied and accepted as an essential part of the layered development of the Semantic Web. Nonmonotonic rule systems, on the other hand, seem also to be a good solution, especially due to their expressive capabilities. The basic motives for using such systems are:

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 [67].

Rules with Exceptions*:* Rules with exceptions are a natural representation for policies and business rules [6]. Priority information is often implicitly or explicitly available to resolve conflicts among rules. Potential applications include security policies [9][57] , business rules [3] , personalization, brokering, bargaining, and automated agent negotiations[39].

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. [44] presents some ideas for possible uses of default inheritance in ontologies.

The following example is used to represent default inheritance in ontologies: Elephants are grey, with the exception of the royal elephants, which are white. We can restate the previous statement by saying that:

- Elephants are grey, except for royal elephants.
- Royal elephants are white.
- All royal elephants are elephants.

By applying a strict form of inheritance we should infer that any instance of the class royal elephant should be grey because it is a subclass of the class elephant. However, we know that the property, color, should be filled with the value, white, for any instance of the class royal elephant. This situation leads naturally to the idea of inheritance by default. We can model inheritance by default by means of non–classical logic. For instance, the above statement can represented in Default Logic as:

$$\frac{elephant(x) : \neg royalElephant(x)}{grey(x)}$$

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.

Some of the mismatches that may occur when someone tries to crate a single ontology by merging two different ontologies with overlapping parts are:

- Same concepts are represented by different names (synonym terms); e.g. term "car" in one ontology and term " automobile " in another ontology.
- The same term is used with different meaning (homonym terms); e.g. term "conductor" has different meaning in music domain than in electrical engineering domain.
- Values in ontologies may be encoded in different formats; e.g. distance may be described as miles or kilometers, or date may be represented as "dd/mm/yyyy" or as " mm-dd-yy"
- Mismatch between part of the domain that is covered by the ontology, or the level of detail to which that domain is modeled, e.g. one ontology might model cars but not trucks. Another one might represent trucks but only classify them into a few categories.

If rule-based ontology languages are used (e.g. DLP [43]) 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 sceptical 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.

3.3 Existing Web Rule Languages

3.3.1 RuleML Markup Language

RuleML [88] is an XML-based markup language that is intended to support rule exchange and interoperation across disparate domains. RuleML allows rules to be expressed as modular components in a declarative way, and uses distinct, standard XML tags to define a rule base, composed of facts and rules.

RuleML encompasses a hierarchy of rules, including reaction rules (event-condition-action rules), transformation rules (functional-equational rules), derivation rules (implicational-inference rules), also specialized to facts ('premiseless' derivation rules) and queries ('conclusionless' derivation rules), as well as integrity-constraints (consistency-maintenance rules).

The RuleML hierarchy of general rules branches into the two direct categories of reaction rules and transformation rules. On the next level, transformation rules specialize to the subcategory of derivation rules. Then, derivation rules have further subsubcategories, namely facts and queries. Finally, queries specialize to integrity constraints. More subdivisions are being worked out, especially for reaction rules. A graphical view of RuleML rules is shown in Figure 3.2.

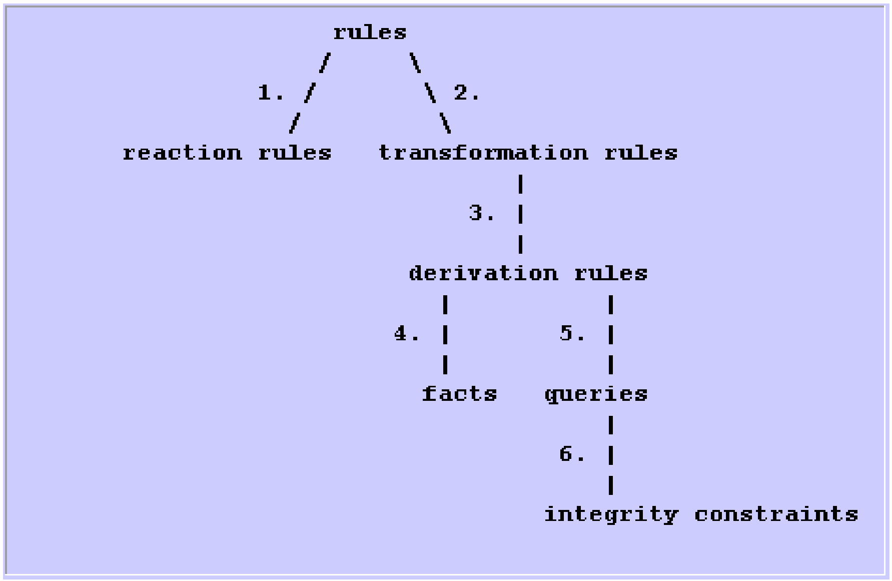


Figure 3.2: The RuleML hierarchy of rules.

Reaction rules can be reduced to general rules that return no value. Transformation rules can be reduced to general rules whose 'event' trigger is always activated.

Derivation rules can be reduced to transformation rules that like characteristic functions on success just return *true*. Facts can be reduced to derivation rules that have an empty (hence, 'true') conjunction of premises. Queries can be reduced to derivation rules that have - similar to refutation proofs - an empty (hence, 'false') disjunction of conclusions or - as in 'answer extraction' - a conclusion that captures the derived variable bindings. Integrity constraints can be reduced to queries that are 'closed' (i.e., produce no variable bindings).

Integrity Rules: Integrity rules, also known as integrity constraints, consist of a logical sentence (in some logical language such as predicate logic or temporal logic). They express assertions that must hold in all evolving states and state transition histories of the discrete dynamic system for which they are defined. The enforcement of constraint rules can be implemented with the help of Event – Condition – Action rules whose event condition refers to state changes that would violate the constraint and whose action would be an alert or some kind of repair action.

Derivation Rules: Derivation rules, in general, consist of one ore more conditions and a conclusion. (Expressions with no condition or no conclusion are called “facts” and “denial constraints”). For specific type of derivation rules, such as definite Horn clauses or normal logic programs, the types of condition and conclusion are specifically restricted. In RuleML, a derivation rule has two roles, _Condition and _Conclusion; the latter being an atomic predicate logic formula and the former a quantifier-free logical formula with weak and strong negation.

Reaction Rules: Reaction rules consist of a mandatory triggering event term, an optional condition, and a triggered action term or a post-condition (or both). While the condition of a reaction rule is, exactly like the condition of a derivation rule, a quantifier-free formula, the post condition is restricted to a conjunction of possibly negated atoms. The post-condition of a reaction rule is either an atomic formula, a negation of an atomic formula or a conjunction of these. Reaction rules can only be applied in the forward direction in a natural fashion, observing / checking events / conditions and performing an action if and when all events / conditions have been perceived or fulfilled. There are basically two types of reaction rules: those that do not have a postcondition, which are the well-known Event-Condition-Action rules, and those that do have a postcondition, which are called ECAP rules.

Transformation Rules: Transformation rules consist of a transformation invoker, a condition, and a transformation return. In RuleML, while these rules generally reuse the same concrete syntax as other rule types, and specifically the condition part, they also introduce new constructs. The <trans> element is the top-level element denoting a transformation rule. It uses the transformation invoker role _headf, optionally followed by a condition role _body, followed by the transformation return role _foot. The element <trans>_headf _body _foot</trans> stands for: “if _body holds then _headf is transformed to _foot”. Without the optional _body part, the transformation is performed unconditionally.

The application direction for each category of rules is: Reaction rules can only be applied in the forward direction in a natural fashion, observing/checking events/conditions and performing an action if and when all events/conditions have been recognized/fulfilled. For transformation rules, on the other hand, the backward direction is normally preferred. Derivation rules can be equally applied in the forward direction as well as in the backward direction, the latter reducing the proof of a goal (conclusion) to proofs of all its subgoals (premises). Since in different situations different application directions of derivation rules may be optimal (forward, backward, or mixed), RuleML does not prescribe any one of these. For facts or 'unit clauses' there is no notion of an application direction. For queries there is the following notion of application direction: as top-down goals, they are proved backward; but they can also be proved forward via 'goal-directed' bottom-up processing. Integrity constraints are usually forward-oriented, i.e. triggered by update events, mainly for efficiency reasons. But they can instead be backward-oriented, trying to show (in)consistency by fulfilling certain conditions (without need for recognizing any event).

The latest XSD version that has been released is RuleML version 0.87. Several rule languages that have recently been developed for the Web, are designed to integrate with RuleML. One of them is SWRL, which is described in the next section.

3.3.2 A Semantic Web Rule Language (SWRL)

SWRL [47] is based on a combination of OWL DL and OWL Lite sublanguages of the OWL Web Ontology Language with the Unary / Binary Datalog RuleML sublanguages of the RuleML Markup Language. It extends the set of OWL axioms to include Horn-like rules. It thus enables Horn-like rules to be combined with an OWL knowledge base. It provides a high-level abstract syntax that extends the OWL abstract syntax described in the OWL Semantics and Abstract Syntax document [75] . It also extends the OWL model-theoretic semantics to provide a formal meaning for OWL ontologies including rules written in this abstract syntax.

The proposed rules are of the form of an implication between an antecedent (body) and consequent (head). The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold. Both the antecedent (body) and consequent (head) consist of zero or more atoms. An empty antecedent is treated as trivially true (*i.e.*, satisfied by every interpretation), so the consequent must also be satisfied by every interpretation; an empty consequent is treated as trivially false (*i.e.*, not satisfied by any interpretation), so the antecedent must also not be satisfied by any interpretation. Multiple atoms are treated as a conjunction. Note that rules with conjunctive consequents could easily be transformed (via the Lloyd-Topor transformations) into multiple rules each with an atomic consequent. The syntax of the rules, as specified by means of EBNF is:

```
rule ::= 'Implies(' { annotation } antecedent consequent ')'
antecedent ::= 'Antecedent(' { atom } ')'
consequent ::= 'Consequent(' { atom } ')'
```

Atoms in these rules can be of the form C(x), P(x,y), sameAs(x,y) or differentFrom(x,y), where C is an OWL description, P is an OWL property, and x,y are either variables, OWL individuals or OWL data values. The syntax used for the atoms, is:

```
atom ::= description '(' i-object ')'
       | individualvaluedPropertyID '(' i-object i-object ')'
       | datavaluedPropertyID '(' i-object d-object ')'
       | sameAs '(' i-object i-object ')'
       | differentFrom '(' i-object i-object ')'
i-object ::= i-variable | individualID
d-object ::= d-variable | dataLiteral
i-variable ::= 'I-variable(' URIreference ')'
d-variable ::= 'D-variable(' URIreference ')'
```

Informally, an atom C(x) holds if x is an instance of the class description C, an atom P(x,y) holds if x is related to y by property P, an atom sameAs(x,y) holds if x is interpreted as the same object as y, and an atom differentFrom(x,y) holds if x and y are interpreted as different objects. Atoms may refer to individuals, data literals, individual variables or data variables. Variables are treated as universally quantified, with their scope limited to a given rule.

An XML syntax is also given for these rules based on RuleML and the OWL XML presentation syntax. The Ontology root element of the OWL XML Presentation Syntax is extended to include "imp" (implication rule) and "var" (variable declaration) axioms as found under the rulebase root of RuleML. Furthermore, an RDF concrete syntax based on the OWL RDF/XML exchange syntax is presented.

3.3.3 TRIPLE

TRIPLE [90] is a rule language for the Semantic Web which is based on Horn logic and borrows many basic features form F-Logic [49] but is especially designed for querying and transforming RDF models. TRIPLE can be viewed as a successor of SiLRI (Simple Logic-based RDF Interpreter [30]). One of the most important differences to F-Logic and SiLRI is that TRIPLE does not have fixed semantics for object-oriented features like classes and inheritance. Its modular architecture allows such features to be defined for object-oriented and other data models like UML, Topic Maps, or RDF Schema. Description logic extensions of RDF (Schema) like OIL, DAML+OIL, and OWL that cannot be fully handled by Horn logic are provided as modules that interact with a description logic classifier, e.g. FaCT [46], resulting in a hybrid rule language. The main features of TRIPLE include:

Namespaces and Resources: TRIPLE has special support for namespaces and resource identifiers. Namespaces are declared via clause-like constructs of the form *nsabbrev* : = *namespace*, e.g.: rdf : = http://www.w3.org/1999/02/22-rdf-syntax-ns#. Resources are written as *nsabbrev:name*, where *nsabrev* is a namespace abbreviation and *name* is the local name of the resource.

Statements and Molecules: Inspired by F-Logic object syntax, an RDF statement (triple) is written as: *subject*[*predicate* → *object*]. Several statements with the same subject can be abbreviated as "molecules":

stefan[*hasAge* → *33; isMarried* → *yes;…*].

RDF statements (and molecules) can be nested, e.g.:

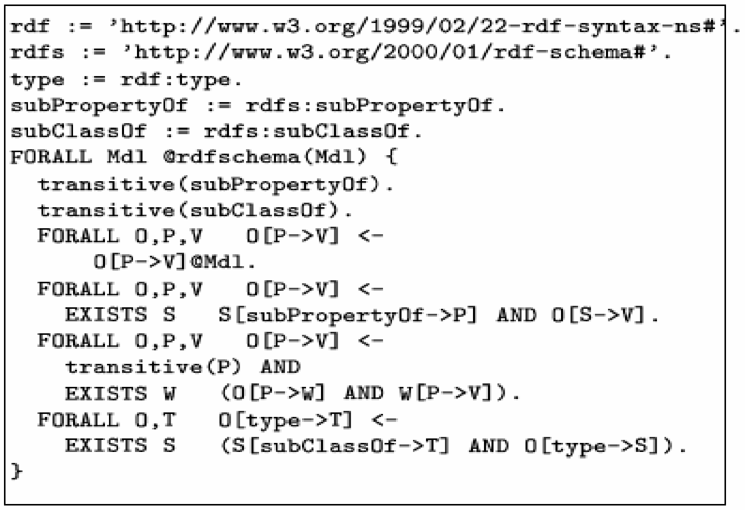
$stefan[marriedTo \rightarrow birgit[hasAge \rightarrow 32]]$.

Models: RDF models, i.e., sets of statements, are made explicit in TRIPLE. Statements, molecules, and also Horn atoms that are true in a specific model are written as *atom@model*, where *atom* is a statement, molecule, or Horn atom and *model* is a model specification.

Logical Formulae: TRIPLE uses the usual set of connectives and quantifiers for building formulae from statements/molecules and Horn atoms, i.e., $\vee, \neg, \forall, \exists$, etc. All variables must be introduced via quantifiers, therefore marking them is not necessary (i.e. TRIPLE does not require variables to start with an uppercase letter as in Prolog).

Clauses and Blocks: A TRIPLE clause is either a fact or a rule. Rule heads may only contain conjunctions of molecules and Horn atoms and must not contain (explicitly or implicitly) any disjunctive or negated expressions. To assert that a set of clauses is true in a specific model, a model block is used: *@model{clauses}*, or if the model specification is parameterized: $\forall$ *Mdl@model(Mdl) {clauses}*.

It is very interesting to see the basic features of RDF Schema, described in the TRIPLE language (Figure 3.3).



```
rdf := 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'.
rdfs := 'http://www.w3.org/2000/01/rdf-schema#'.
type := rdf:type.
subPropertyOf := rdfs:subPropertyOf.
subClassOf := rdfs:subClassOf.
FORALL Mdl @rdfschema(Mdl) {
  transitive(subPropertyOf).
  transitive(subClassOf).
  FORALL O,P,V   O[P->V] <-
      O[P->V]@Mdl.
  FORALL O,P,V   O[P->V] <-
    EXISTS S   S[subPropertyOf->P] AND O[S->V].
  FORALL O,P,V   O[P->V] <-
    transitive(P) AND
    EXISTS W   (O[P->W] AND W[P->V]).
  FORALL O,T   O[type->T] <-
    EXISTS S   (S[subClassOf->T] AND O[type->S]).
}
```

Figure 3.3: RDF Schema in TRIPLE

3.4 Choice of Formalism

After presenting some existing web rule languages in the previous section, and several approaches to nonmonotonic reasoning in Chapter 2, in this section we justify the choice of Defeasible Logic as the rule language for our system. The main reasons are basically three:

a) Defeasible logic offers more reasoning capabilities than the other existing web rule languages, as it can reason with both monotonic and nonmonotonic rules. SWRL, TRIPLE and most of the other existing rule languages that have been designed for web applications are based on monotonic rule systems. Nonmonotonic rule systems offer more expressive capabilities, allowing to reason with incomplete and inconsistent information, and with rules with exceptions. In section 3.2.1 we described the reasons why such systems should also be used in the develpoment of web applications.

b) Defeasible logic is preferred to the classical approaches of nonmonotonic reasoning as default logic, autoepistemic logic etc., as it embodies the concept of preference. Preferences helps to resolve possible conflicts between rules of a theory, and add to the expressive power of the rule language. The sources of information about preference and the importance of preference in web application are described in detail in Section 2.5. In the same section, we refer to some extensions of the classical nonmonotonic logics, which are equipped with preferences. However these extensions add more computational complexity to the logics, making them unsuitable for real-world web applications.

c) Compared to other nonmonotonic logics, defeasible logic has the additional very important advantage of its relatively low computational complexity, making it preferable for applications that use very large rule sets. According to Maher [62], inference in propositional defeasible logic has linear complexity, which is much lower than the computational complexity of sceptical default reasoning, sceptical autoepistemic reasoning and propositional circumscription, which is $\prod_2^p$ - complete.

4 Defeasible Logic

In Chapter 2, we presented the basic characteristics of Defeasible Logic, as it was introduced by Donald Nute [73]. In this chapter, we give a more formal definition, a proof theory and some theorems concerning Defeasible Logic. At the end of the chapter we present two different variables, the ambiguity blocking and the ambiguity propagation variant, and we describe an additional feature, the use of conflicting literals.

4.1 Formal Definition

In this report we restrict attention to essentially propositional Defeasible Logic. Rules with free variables are interpreted as rule schemas, that is, as the set of all ground instances. If q is a literal $\sim q$ denotes the complementary literal (if q is a positive literal p then $\sim q$ is $\neg p$; and if q is $\neg p$, then $\sim q$ is p).

Rules are defined over a *language* (or *signature*) Σ, the set of propositions (atoms) and labels that may be used in the rule. In cases where it is unimportant to refer to the language of D, Σ will not be mentioned.

A *rule* r: $A(r) \hookrightarrow C(r)$ consists of its unique *label* r, its *antecedent* $A(r)$ ($A(r)$ may be omitted if it is the empty set) which is a finite set of literals, an arrow $\hookrightarrow$ (which is a placeholder for concrete arrows to be introduced in a moment), and its *head* (or *consequent*) $C(r)$ which is a literal. In writing rules we omit set notation for antecedents, and sometimes we omit the label when it is not relevant for the context. There are three kinds of rules, each represented by a different arrow. Strict rules use $\rightarrow$, defeasible rules use $\Rightarrow$, and defeaters use $\leadsto$.

Given a set R of rules, we denote the set of all strict rules in R by Rs, the set of strict and defeasible rules in R by Rsd , the set of defeasible rules in R by Rd , and the set of defeaters in R by $Rdft$. $R[q]$ denotes the set of rules in R with consequent q.

A *superiority relation on* R is a transitive relation $>$ on R. When $r1 > r2$, then $r1$ is called *superior* to $r2$, and $r2$ *inferior* to $r1$. Intuitively, $r1 > r2$ expresses that $r1$ overrules $r2$, should both rules be applicable. Typically we assume $>$ to be acyclic (that is, the transitive closure of $>$ is irreflexive).

A *defeasible theory* D is a triple $(F, R, >)$ where F is a finite set of literals (called *facts*), R a finite set of rules, and $>$ an acyclic superiority relation on R. D is called *decisive* if the atom dependency graph of D is acyclic.

4.2 Proof Theory

A *conclusion* of D is a tagged literal and can have one of the following four forms:

- $+\Delta\, q$ which is intended to mean that q is definitely provable in D.
- $-\Delta\, q$ which is intended to mean that we have proved that q is not definitely provable in D.
- $+\partial\, q$ which is intended to mean that q is defeasibly provable in D.
- $-\partial\, q$ which is intended to mean that we have proved that q is not defeasibly provable in D.

If we are able to prove q definitely, then q is also defeasibly provable. This is a direct consequence of the formal definition below. It resembles the situation in, say, default logic: a formula is sceptically provable from a default theory $T = (W, D)$ (in the sense that it is included in each extension) if it is provable from the set of facts W.

Provability is defined below (Figure 4.1). It is based on the concept of a *derivation* in $D = (F,R,>)$. A derivation is a finite sequence $P = (P(1),\ldots,P(n))$ of tagged literals satisfying the following conditions ($P(1..i)$ denotes the initial part of the sequence P of length i):

```
+Δ : If $P(i+1) = +\Delta\, q$ then either
        $q \in F$ or
        $\exists r \in R_s[q]\ \forall a \in A(r): +\Delta\, a \in P(1..i)$.
```

Figure 4.1: Definite Provability in Defeasible Logic.

That means, to prove $+\Delta\, q$ we need to establish a proof for q using facts and strict rules only. This is a deduction in the classical sense — no proofs for the negation of q need to be considered (in contrast to defeasible provability below, where opposing chains of reasoning must be taken into account, too).

To prove $-\Delta\, q$, i.e., that q is not definitely provable, q must not be a fact. In addition, we need to establish that every strict rule with head q is *known to be* inapplicable. Thus for every such rule *r* there must be at least one antecedent *a* for which we have established that *a* is not definitely provable ($-\Delta\, a$, Figure 4.2).

```
−Δ: If $P(i + 1) = -\Delta\, q$ then
        $q \notin F$ and
        $\forall r \in R_s[q]\ \exists a \in A(r): -\Delta\, a \in P(1..i)$.
```

Figure 4.2: Definite Non-provability in Defeasible logic.

It is worth noticing that this definition of nonprovability does not involve loop detection. Thus if *D* consists of the single rule $p \rightarrow p$, we can see that *p* cannot be proven, but Defeasible Logic is unable to prove $-\Delta\, p$.

```
+∂: If $P(i + 1) = +\partial q$ then either
        (1) $+\Delta\, q \in P(1..i)$ or
        (2) (2.1) $\exists r \in R_{sd}[q]\ \forall a \in A(r): +\partial a \in P(1..i)$ and
            (2.2) $-\Delta\, {\sim}q \in P(1..i)$ and
            (2.3) $\forall s \in R[{\sim}q]$ either
                (2.3.1) $\exists a \in A(s): -\partial a \in P(1..i)$ or
                (2.3.2) $\exists t \in R_{sd}[q]$ such that
                        $\forall a \in A(t): +\partial a \in P(1..i)$ and $t > s$.
```

Figure 4.3: Defeasible Provability in Defeasible Logic.

To show that *q* is provable defeasibly (Figure 4.3) we have two choices: (1) We show that *q* is already definitely provable; or (2) we need to argue using the defeasible part of *D* as well. In particular, we require that there must be a strict or defeasible rule with head *q* which can be applied (2.1). But now we need to consider possible "attacks", i.e., reasoning chains in support of *~q*. To be more specific: to prove *q* defeasibly we must show that *~q* is not definitely provable (2.2). Also (2.3) we must consider the set of all rules which are not known to be inapplicable and which have head *~q*. Essentially each such rule *s* attacks the conclusion *q*. For *q* to be provable, each such rule *s* must be counterattacked by a rule *t* with head *q* with the following properties: (i) *t* must be applicable at this point, and (ii) *t* must be stronger than *s*. Thus each attack on the conclusion *q* must be counterattacked by a stronger rule.

The definition of the proof theory of Defeasible Logic is completed by the condition $-\partial$ (Figure 4.4). It is nothing more than a strong negation of the condition $+\partial$.

```
$-\partial$: If $P(i+1) = -\partial q$ then
        (1) $-\Delta q \in P(1..i)$ and
        (2) (2.1) $\forall r \in R_{sd}[q]\ \exists a \in A(r): -\partial a \in P(1..i)$ or
            (2.2) $+\Delta \sim q \in P(1..i)$ or
            (2.3) $\exists s \in R[\sim q]$ such that
                (2.3.1) $\forall a \in A(s): +\partial a \in P(1..i)$ and
                (2.3.2) $\forall t \in R_{sd}[q]$ either
                        $\exists a \in A(t): -\partial a \in P(1..i)$ or $t \not> s$.
```

Figure 4.4: Defeasible Non-provability in Defeasible Logic.

To prove that *q* is not defeasibly provable, we must first establish that it is not definitely provable. Then we must establish that it cannot be proven using the defeasible part of the theory. There are three possibilities to achieve this: either we have established that none of the (strict and defeasible) rules with head *q* can be applied (2.1); or *~q* is definitely provable (2.2); or there must be an applicable rule *s* with head *~q* such that no possibly applicable rule *t* with head *q* is superior to *s* (2.3).

The elements of a derivation *P* in *D* are called *lines* of the derivation. We say that a tagged literal *L* is *provable* in *D* = (*F*, *R*, >), denoted $D \vdash L$, iff there is a derivation in *D* such that *L* is a line of *P*. When *D* is obvious from the context we write $\vdash L$.

It is instructive to consider the conditions $+\partial$ and $-\partial$ in the terminology of *teams*, borrowed from Grosof [42]. At some stage there is a team *A* consisting of the applicable rules with head *q*, and a team *B* consisting of the applicable rules with head *~q*. These teams compete with one another. Team *A* wins iff every rule in team *B* is overruled by a rule in team *A*; in that case we can prove $+\partial q$. Another case is that team *B* wins, in which case we can prove $+\partial{\sim}q$. But there are several intermediate cases, for example one in which we can prove that neither *q* nor *~q* are provable. And there are cases where nothing can be proved (due to loops).

Proposition 1. [14] *If D is decisive, then for each literal p:*

(a) either $D \vdash +\Delta p$ *or* $D \vdash -\Delta p$

(b) either $D \vdash -\partial p$ *or* $D \vdash +\partial p$

Not every defeasible theory satisfies this property. For example, in the theory consisting of the single rule $p \Rightarrow p$ neither $-\partial p$ nor $+\partial p$ is provable. The proof of the proposition can be found in [14].

Proposition 2. [5]*Consider a defeasible theory D.*

(1) If $D \nvdash -\Delta{\sim}p$ *and* $D \nvdash +\Delta p$ *then* $D \nvdash +\partial p$.

(2) *If* $D \vdash +\Delta{\sim}p$ *and* $D \vdash -\Delta p$ *then* $D \vdash -\partial p$.

(3) *If* $D \vdash +\partial{\sim}p$ *and* $D \vdash -\Delta p$ *and* $D \nvdash -\partial p$ *then D is cyclic.*

Theorem 3. [5] *If D is an acyclic defeasible theory, then D is conclusion equivalent to a theory D' that contains no use of the superiority relation, nor defeaters. If D is a cyclic defeasible theory, then D is conclusion equivalent to a theory D' that contains no use of defeaters, and if D' contains cycles then they have length 2, and each cycle involves the only two rules for a literal and its complement.*

Proposition 4. [5] *Let D be an acyclic defeasible theory.*

If $D \vdash +\partial p$ *and* $D \vdash +\partial{\sim}p$ *then* $D \vdash +\Delta p$ *and* $D \vdash +\Delta{\sim}p$.

Consequently, if D contains no strict rules and no facts and $D \vdash +\partial q$, *then* $D \vdash -\partial{\sim}q$.

The proves for the Propositions 2, 4 and for the Theorem 3 can be found in [5].

Governatori *et. al* [40] describe Defeasible Logic and its variants in argumentation-theoretic terms. Under the argumentation semantics, proof trees are grouped together as arguments, and conflicting arguments are resolved by notions of argument defeat that reflect defeat in defeasible logic.

Maher [61] gives a denotational-style semantics to Defeasible Logic in , providing another useful analysis of this logic. The semantics is compositional, and fully abstract in all but one syntactic class.

A model-theoretic semantics semantics of Defeasible Logic is given by Maher in [63]. This semantics follows Nute's semantics for LDR [72] in that models represent a state of mind or "belief state" in which definite knowledge (that which is "known") is distinguished from defeasible knowledge (that which is “believed”). A major difference from [72] is that adopts partial models as the basic from which to work.

4.3 Ambiguity Blocking and Ambiguity Propagating Behavior

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

Example: Consider the defeasible theory from Chapter

r_1: *quaker(X)* $\Rightarrow$ *pacifist(X)*
r_2: *republican(X)* $\Rightarrow$ *~pacifist(X)*
r_3: *pacifist(X)* $\Rightarrow$ *~hasGun(X)*
r_4: *livesInChicago(X)* $\Rightarrow$ *hasGun(X)*
quaker(a)
republican(a)
livesInChicago(a)
$r_3 > r_4$

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

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

In the following we present the proof theory for the ambiguity propagation variant of defeasible logic. The first step is to determine when a literal is “supported” in a defeasible theory D (Figure 4.5). Support for a literal *p* ($+\Sigma\, p$) consists of a monotonic chain of reasoning that would lead us to conclude p in the absence of conflicts. This leads to the following inference conditions:

$+\Sigma$: If $P(i+1) = +\Sigma q$ then either

 (1) $p \in F$ or

 (2) $\exists r \in R_{sd}[q]\ \forall a \in A(r): +\Sigma a \in P(1..i)$.

$-\Sigma$: If $P(i+1) = -\Sigma q$ then

 (1) $p \notin F$ and

 (2) $\forall r \in R_{sd}[q]\ \exists a \in A(r): -\Sigma a \in P(1..i)$.

Figure 4.5: Supporting Literals in Defeasible Logic.

A literal that is defeasibly provable is supported, but a literal may be supported even though it is not defeasibly provable. Thus support is a weaker notion than defeasible provability. For example, given two rules $\Rightarrow q$ and $\Rightarrow\sim q$, both q and $\sim q$ are supported, but neither is defeasibly provable.

We can achieve ambiguity propagation behavior by making a minor change to the inference condition for $+\partial$: instead of requiring that every attack on p be inapplicable in the sense of $-\partial$, now we require that the rule for ~p be inapplicable because one of its antecedents cannot be *supported.* By making attack easier we are imposing a stronger condition for proving a literal defeasibly. Here is the formal definition:

$+\partial_{ap}$: If $P(i+1) = +\partial q_{ap}$ then either

 (1) $+\Delta q \in P(1..i)$ or

 (2) (2.1) $\exists r \in R_{sd}[q]\ \forall a \in A(r): +\partial_{ap} a \in P(1..i)$ and

  (2.2) $-\Delta \sim q \in P(1..i)$ and

  (2.3) $\forall s \in R[\sim q]$, either

    (2.3.1) $\exists a \in A(s): -\Sigma a \in P(1..i)$ or

    (2.3.2) $\exists t \in R_{sd}[q]$ such that

      $\forall a \in A(t): +\partial_{ap} a \in P(1..i)$ and $t > s$.

$-\partial_{ap}$: If $P(i+1) = -\partial q_{ap}$ then

 (1) $-\Delta q \in P(1..i)$ and

 (2) (2.1) $\forall r \in R_{sd}[q]\ \exists a \in A(r): -\partial_{ap} a \in P(1..i)$ or

  (2.2) $+\Delta \sim q \in P(1..i)$ or

  (2.3) $\exists s \in R[\sim q]$ such that

    (2.3.1) $\forall a \in A(s): +\Sigma a \in P(1..i)$ and

    (2.3.2) $\forall t \in R_{sd}[q]$ either

      $\exists a \in A(t): -\partial a_{ap} \in P(1..i)$ or $t \not> s$.

Figure 4.6: Defeasible provability in the Ambiguity Propagating Variant of Defeasible Logic.

A preference for ambiguity blocking or ambiguity propagating behavior is one of the properties of nonmonotonic inheritance nets over which intuitions can clash [95]. 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.

4.4 Conflicting Literals

So far only conflicts among rules with complementary heads were detected 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: $q_1,q_2,\ldots,q_n \rightarrow \sim L$ *for all rules* r_i: $q_1,q_2,\ldots,q_n \rightarrow M$

r_i: $q_1,q_2,\ldots,q_n \rightarrow \sim M$ *for all rules* r_i: $q_1,q_2,\ldots,q_n \rightarrow L$

r_i: $q_1,q_2,\ldots,q_n \Rightarrow \sim L$ *for all rules* r_i: $q_1,q_2,\ldots,q_n \Rightarrow M$

r_i: $q_1,q_2,\ldots,q_n \Rightarrow \sim L$ *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 :

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

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

$r_1 > r_2'$ $\quad r_1' > r_2$

4.5 Exisiting Defeasible Logic Implementations

4.5.1 d-Prolog

d-Prolog [28] is a query-answering interpreter for defeasible logic implemented in about 300 lines of Prolog. Its intended input is mostly small, non-recursive inheritance problems. The strict rules are implemented directly as Prolog rules. Unfortunately, the d-Prolog implementation of defeasible logic is flawed. The interpreter follows the Prolog computation rule and consequently has the same incompleteness that Prolog has. It cannot deal with theories that contain cyclic dependencies, so we could not test the system in the case of cyclic theories. Other deficiencies of d-Prolog, compared to our system are that it does not implement the ambiguity propagation variant of defeasible logic, and does not deal with user-defined conflicting literals. Finally, it does not integrate with Semantic Web (it does not treat RDF data, nor does it use an XML-based / RDF-based syntax).

4.5.2 Deimos

Deimos [64] is a flexible, query processing system based on Haskell. It does not support variables, and accepts only propositional rules. It implements several variants, but not conflicting literals. The centre of the system is the prover. It implements backward-chaining theorem prover for defeasible logic based almost directly on the inference rules. The system also includes a program (*DTScale*) that generates the scalable theories used as test case in our experimental evaluation. Deimos performs a depth-first search, with memorization and loop-checking, for a proof in defeasible logic. Memorization allows the system to recognize that a conclusion has already been proved (or disproved), while loop checking also detects when a conclusion occurs twice in a branch of the search tree. Loop-checking is necessary for the depth-first search to be complete, whereas memorization is purely a matter of efficiency. Deimos does not also integrate with Semantic Web. Comparing to our system, it has the additional feature that it also treats defeaters.

4.5.3 DR-Device

This is another effort on implementing defeasible reasoning, albeit with a different approach. DR-DEVICE [98] is implemented in Jess, and integrates well with RuleML and RDF. It is a system for query answering. Its architecture is based on the rule system CLIPS, and is in fact an extension of R-Device: a system for rules on RDF data. The rules are expressed using the rule language of CLIPS, or the OO-RuleML syntax. Compared to our system, DR-DEVICE supports only one variant, ambiguity blocking, and does not also treat user-defined conflicting literals, thus it does not offer the flexibility of our implementation. Moreover, it is based on a translation of defeasible theories into a non-logical language, and thus lacks in declarativity.

5 Translation into Logic Programs

There are two approaches for translating defeasible theories into logic programs. The first one uses a meta-program to express defeasible logic in logic programming terms. This approach was followed in various studies, including [20]. The second approach, upon which we base our work, is more direct, and was first presented in [7]. In general, for the translation of defeasible theories into logic programs, two logic program semantics can be used: Kunen semantics and Well-Founded semantics. The latter has the advantage that it can detect cycles in the theories (it does not run into infinite loops). Under the second approach, 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* ⇔

p is included in the Well-Founded model of *P(D)*

To achieve this goal, the rules of a defeasibly theory *D* are translated into logic program clauses, using control literals that carry meaning regarding the applicability status of the rules. The features of Defeasible Logics which are supported by that translation are: facts, strict rules, and defeasible rules that may support conflicting conclusions. 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.

5.1 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,p):- ok'(r,*s_1*,p),…,ok'(r,*s_m*,p).*

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

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

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

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

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

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 *¬p.*
- $d_3(r,s_i,p)$ says that it is ok to apply *r* w.r.t. s_i is blocked.
- $d_4(r,s_i,p)$ says that it is ok to apply *r* w.r.t. s_i is blocked.
- $d_5(r,q_i,p)$ specifies the only way a rule can be blocked: it must be impossible to prove one of its antecedents.

[1] For the implementation of the translation, we use *sk_not* as the negation operator. The use of this operator is described in Chapter 6.

- *$d_6(r,s_i,p)$* 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), $d_1(r)$, $d_2(r)$, $d_3(r,s_i,p)$, $d_4(r,s_i,p)$, $d_5(r,q_i,p)$, $d_6(r,s_i,p)$, e(r,s).*

Example: Consider the defeasible theory *D* from the example of Chapter 4. Assuming ambiguity blocking behavior, *D* is translated into the logic program *P(D)*, which consists of the following clauses:

r1	defeasibly(pacifist(X)):- defeasibly(quaker(X)),sk_not definitely(~(pacifist(X))). ok(r1,pacifist(X)):- okk(r1,r2,pacifist(X)). okk(r1,r2,pacifist(X)):- blocked(r2,~(pacifist(X))). okk(r1,r2,pacifist(X)):- defeated(r2,~(pacifist(X))). defeated(r1,pacifist(X)):- sk_not blocked(r2,~(pacifist(X))),sup(r1,r2). blocked(r1,pacifist(X)):- sk_not defeasibly(quaker(X)).
r2	defeasibly(~(pacifist(X))):- defeasibly(republican(X)),sk_not definitely(pacifist(X)),ok(r2,~(pacifist(X))). ok(r2,~(pacifist(X))):- okk(r2,r1,~(pacifist(X))). okk(r2,r1,~(pacifist(X))):- blocked(r1,pacifist(X)). okk(r2,r1,~(pacifist(X))):- defeated(r1,pacifist(X)). defeated(r2,~(pacifist(X))):- sk_not blocked(r1,pacifist(X)),sup(r1,r2). blocked(r2,~(pacifist(X))):- sk_not defeasibly(republican(X)).
r3	defeasibly(~(hasGun(X))):- defeasibly(pacifist(X)),sk_not definitely(hasGun(X)),ok(r3,~(hasGun(X))). ok(r3,~(hasGun(X))):- okk(r3,r4,~(hasGun(X))). okk(r3,r4,~(hasGun(X))):- blocked(r4,hasGun(X)). okk(r3,r4,~(hasGun(X))):- defeated(r4,hasGun(X)). defeated(r3,~(hasGun(X))):- sk_not blocked(r4,hasGun(X)),sup(r4,r3). blocked(r3,~(hasGun(X))):- sk_not defeasibly(pacifist(X)).
r4	defeasibly(hasGun(X)):- defeasibly(livesInChicago(X)),sk_not definitely(~(hasGun(X))),ok(r4,hasGun(X)). ok(r4,hasGun(X)):- okk(r4,r3,hasGun(X)). okk(r4,r3,hasGun(X)):- blocked(r3,~(hasGun(X))). okk(r4,r3,hasGun(X)):- defeated(r3,~(hasGun(X))). defeated(r4,hasGun(X)):- sk_not blocked(r3,~(hasGun(X))),sup(r3,r4). blocked(r4,hasGun(X)):- sk_not defeasibly(livesInChicago(X)).
	definitely(quaker(a)). defeasibly(quaker(a)):- definitely(quaker(a)).
	definitely(republican(a)). defeasibly(republican(a)):- definitely(republican(a)).
	definitely(livesInChicago(a)). defeasibly(livesInChicago(a)):- definitely(livesInChicago(a)).
	sup(r3,r4).

Under the above translation, *defeasibly(hasGun(a))* is included in the answer set of the logic program *P(D).* This comes in agreement with the ambiguity blocking behavior of the defeasible theory *D*, according to which, a rule with ambiguous premises cannot fire, and override the conflicting rules (even if it superior to them). So the ambiguity of a literal is blocked, and is not propagated to its dependent conclusions. Thus, this logic program corresponds exactly to the ambiguity blocking behavior of the defeasible theory *D*.

5.2 Translation of Ambiguity Propagation Behavior

We must make some changes to the procedure of the translation that we describe 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 $d_1(r)$, $d_2(r)$, $d_4(r,s_i,p)$, $d_5(r,q_i,p)$, $d_6(r,s_i,p)$, *e(r,s)* also remain the same. In order to support the ambiguity propagation behavior, we must change $d_3(r,s_i,p)$ 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:

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

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

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

$d_7(r,q_i,p)$: *obstructed(r,p):- 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,p).*

- $d_3'(r,s_i,p)$ says that it is ok to apply *r* w.r.t. s_i is obstructed.
- $d_7(r,q_i,p)$ 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 *$a(p)$, $b(r)$, $c(p)$, $d_1(r)$, $d_2(r)$, $d_3'(r,s_i,p)$, $d_4(r,s_i,p)$, $d_5(r,q_i,p)$, $d_6(r,s_i,p)$, $d_7(r,q_i,p)$, $d_8(r)$, $e(r,s)$*.

Example: Consider the defeasible theory *D* from the example of Chapter 4. Assuming ambiguity propagation behavior, *D* is translated into the logic program *P(D)*, which consists of the following clauses:

r1	`defeasibly(pacifist(X)):- defeasibly(quaker(X)),sk_not definitely(~(pacifist(X))).`
	`ok(r1,pacifist(X)):- okk(r1,r2,pacifist(X)).`
	`okk(r1,r2,pacifist(X)):- obstructed(r2,~(pacifist(X))).`
	`okk(r1,r2,pacifist(X)):- defeated(r2,~(pacifist(X))).`
	`defeated(r1,pacifist(X)):- sk_not blocked(r2,~(pacifist(X))),sup(r1,r2).`
	`blocked(r1,pacifist(X)):- sk_not defeasibly(quaker(X)).`
	`obstructed(r1,pacifist(X)):- sk_not supported(quaker(X)).`
	`supported(pacifist(X)):- supported(quaker(X)),sk_not defeated(r1,pacifist(X)).`
r2	`defeasibly(~(pacifist(X))):- defeasibly(republican(X)),sk_not definitely(pacifist(X)),ok(r2,~(pacifist(X))).`
	`ok(r2,~(pacifist(X))):- okk(r2,r1,~(pacifist(X))).`
	`okk(r2,r1,~(pacifist(X))):- obstructed(r1,pacifist(X)).`
	`okk(r2,r1,~(pacifist(X))):- defeated(r1,pacifist(X)).`
	`defeated(r2,~(pacifist(X))):- sk_not blocked(r1,pacifist(X)),sup(r1,r2).`
	`blocked(r2,~(pacifist(X))):- sk_not defeasibly(republican(X)).`
	`obstructed(r2,~(pacifist(X))):- sk_not supported(republican(X)).`
	`supported(~(pacifist(X))):- supported(republican(X)),sk_not defeated(r2,~(pacifist(X))).`
r3	`defeasibly(~(hasGun(X))):- defeasibly(pacifist(X)),sk_not definitely(hasGun(X)),ok(r3,~(hasGun(X))).`
	`ok(r3,~(hasGun(X))):- okk(r3,r4,~(hasGun(X))).`
	`okk(r3,r4,~(hasGun(X))):- obstructed(r4,hasGun(X)).`
	`okk(r3,r4,~(hasGun(X))):- defeated(r4,hasGun(X)).`
	`defeated(r3,~(hasGun(X))):- sk_not blocked(r4,hasGun(X)),sup(r4,r3).`
	`blocked(r3,~(hasGun(X))):- sk_not defeasibly(pacifist(X)).`
	`obstructed(r3,~(hasGun(X))):- sk_not supported(pacifist(X)).`
	`supported(~(hasGun(X))):- supported(pacifist(X)),sk_not defeated(r3,~(hasGun(X))).`
r4	`defeasibly(hasGun(X)):- defeasibly(livesInChicago(X)),sk_not definitely(~(hasGun(X))),ok(r4,hasGun(X)).`
	`ok(r4,hasGun(X)):- okk(r4,r3,hasGun(X)).`
	`okk(r4,r3,hasGun(X)):- obstructed(r3,~(hasGun(X))).`
	`okk(r4,r3,hasGun(X)):- defeated(r3,~(hasGun(X))).`
	`defeated(r4,hasGun(X)):- sk_not blocked(r3,~(hasGun(X))),sup(r3,r4).`
	`blocked(r4,hasGun(X)):- sk_not defeasibly(livesInChicago(X)).`
	`obstructed(r4,hasGun(X)):- sk_not supported(livesInChicago(X)).`
	`supported(hasGun(X)):- supported(livesInChicago(X)),sk_not defeated(r4,hasGun(X)).`
	`definitely(quaker(a)).`
	`defeasibly(quaker(a)):- definitely(quaker(a)).`
	`supported(quaker(a)):- definitely(quaker(a)).`

	`definitely(republican(a)).`
	`defeasibly(republican(a)):- definitely(republican(a)).`
	`supported (republican(a)):- definitely(republican(a)).`

	`definitely(livesInChicago(a)).`
	`defeasibly(livesInChicago(a)):- definitely(livesInChicago(a)).`
	`supported (livesInChicago(a)):- definitely(livesInChicago(a)).`

	`sup(r3,r4).`

Under this translation, neither *defeasibly(hasGun(a))*, nor *defeasibly(~hasGun(a)))* is included in the answer set of the logic program, agreeing with the defeasible propagation behavior of the defeasible theory *D*, under which no conclusion can be drawn about literals, which depend on ambiguous premises. *P(D)* corresponds exactly to the ambiguity propagation variant of *D*.

5.3 Properties of the Translation

The size of the logic program *P(D)* which corresponds to a defeasible theory *D* depends on the number of facts, rules and superiority relations of the defeasible theory *D*, but also on the behavior of *D*. In general, theories which consist of sets of defeasible rules that attack each other, lead to programs with much more clauses than theories with rules that do not support conflicting conclusions. The ambiguity propagation variable of a defeasible theory *D* leads to a logic program with up to 30% more clauses in comparison with the program that corresponds to the ambiguity blocking variant of the same theory.

Proposition 1. *The size of the logic program P(D) which corresponds to the ambiguity blocking variant of a defeasible theory D is bound by* $2\times f + 2\times s + (4+d)\times d$*, where f is the number of facts or strict rules of the theory, s is the number of superiority relations and d is the number of defeasible rules.*

According to Proposition 1, a theory with ten defeasible rules, five of them supporting a conclusion *p*, and the other five supporting *~p*, is translated into a logic program with 140 clauses (each defeasible rule is translated into 14 program clauses). If the same rules do not support conflicting conclusions, the size of the logic program is 10, which is the size of the defeasible theory (in the latter case each rule of the defeasible theory is translated into a single program clause).

Proposition 2. *The size of the logic program P(D) which corresponds to the ambiguity propagation variant of a defeasible theory D is bound by* $2\times f + 2\times s +$

(6+d) ×d, where f is the number of facts or strict rules of the theory, s is the number of superiority relations and d is the number of defeasible rules.

According to Proposition 2, the same theory that we describe above is translated to a logic program with 160 program clauses. In this case each rule of the defeasible theory is translated into 16 clauses of the logic program.

For the implementation of the translation, we used *sk_not* as the negation operator. This is the negation operator of XSB [102] (the logic programming tool that we used), which allows for the correct execution of programs with well founded semantics. Under the well-founded model, the relationship between a defeasible theory *D* and its translation *P(D)* can be described with the following theorem.

Theorem 3 [5].

(a). $D \vdash +\Delta p \Leftrightarrow$ *definitely(p) is included in the well-founded model of P(D).*

(b). $D \vdash -\Delta p \Leftrightarrow$ *sk_not definitely(p) is included in the well-founded model of P(D).*

(c). $D \vdash +\partial p \Leftrightarrow$ *defeasibly(p) is included in the well-founded model of P(D).*

(d). $D \vdash -\partial p \Leftrightarrow$ *sk_not defeasibly(p) is included in the well-founded model of P(D).*

6 Implementation Architecture

6.1 Overview of the Architecture

Our goal is to develop a system that supports not only the basics of defeasible logic, but also the two different variants (ambiguity blocking and ambiguity propagation) of this logic, and the use of conflicting literals. In Figure 6.1 we present the overall architecture of our system.

The system works in the following way: The user imports defeasible theories, either using the syntax of defeasible logic (described in Chapter 4), or in the RuleML [88] syntax, that we describe below. The former theories are checked by the DL Parser, and if they are syntactically correct, they are passed to the Logic Translator, which

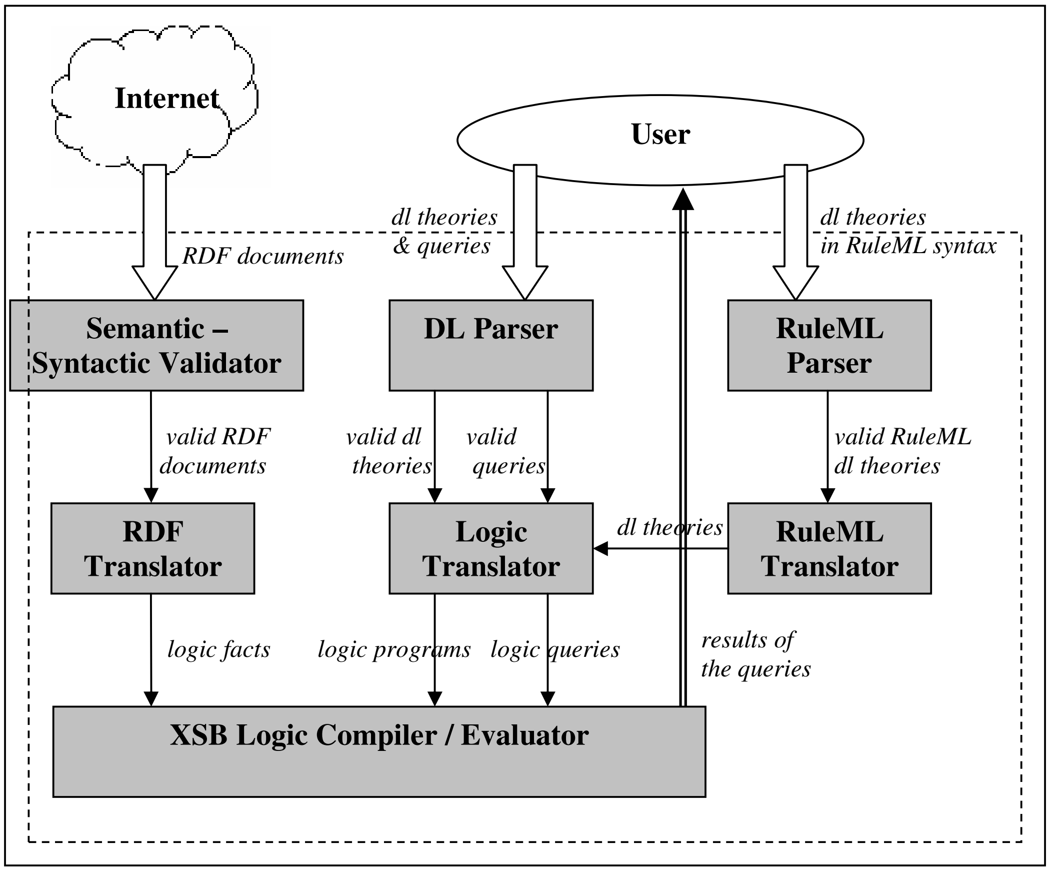


Figure 6.1: The Overall Architecture of our System.

translates them into logic programs. The RuleML defeasible theories are checked by the RuleML Parser and translated into defeasible theories, which are also passed to the Logic Translator and transformed into logic programs. XSB [102], which is the logic programming system that we use, compiles the logic programs, and evaluates the answers to the user's queries. The queries are expressed in a standard syntax (that we describe below). They are checked by the DL Parser, and translated into Prolog queries, before being applied to the compiled programs. An additional functionality that the system supports, is that it has the ability to treat RDF data as facts of the user's defeasible theories. The RDF data are retrieved from the Web, and validated by the Semantic – Syntactic Validator [101] , before being loaded to the system. The system employs the SWI RDF parser [91] to load the valid RDF data and translate them into RDF triples. The triples are then translated into Prolog facts, which are passed to the Logic Compiler. Below, we describe in more detail, each of the tools mentioned above.

6.2 The DL Parser

The parser is responsible for parsing the user's defeasible theory (expressed in the defeasible logic syntax), and for checking whether the theory is syntactically correct. The theory is considered to be correct, if it follows the standard syntax of defeasible logic, as described in Chapter 4. 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 correct, the parser creates a symbol table, which contains all the facts, rules and superiority relations of the user's defeasible theory. The symbol is 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 Chapter 4.

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 *p* is not proved strictly?
- *+d p* :is it concluded that *p* is proved defeasibly?

- *-d p* : is it concluded that *p* not proved defeasibly?

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

The source code of the DL Parser is written in C. For the creation of the code, we employed *lex&yacc* [56].

6.3 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 Chapter 5. The only difference is that, instead of *not* we use *sk_not*, which is XSB's negation operator and allows for the correct execution of programs with well founded semantics. By using this operator, we allow the system to deal with cyclic theories. 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).

The source code of the Logic Translator is written in C. For the creation of the code, we employed *lex&yacc* [56].

6.4 The RuleML Parser & RuleML Translator

Another part of our work is the creation of a DTD which allows to translate XML files to defeasible theories and vice versa. This DTD is in fact an extension of the RuleML DTDs [88]. It covers both strict and defeasible rules, as well as the superiority relations between these rules. The elements of the RuleML DTD that we add / modify 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 superiority relation between these two rules.

In Figure 6.2, 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)>
```

Figure 6.2: The modified RuleML DTD for the defeasible theories

In order to give the user the ability to create defeasible theories using the RuleML syntax, we implemented the RuleML Parser, and the RuleML Translator. The former parses the RuleML defeasible theories and, if they are syntactically correct, passes the theories to the Translator. The role of the RuleML Translator is to translate them into the defeasible logic syntax, according to the DTD we described above. The defeasible theories are then passed to the Logic Translator, which translates them into logic programs. The source code of both tools is written in C. For the creation of the code, we employed lex&yacc [56].

6.5 The Logic Compiler & Evaluator

The role of the Logic Compiler is to compile the logic program *P(D)*, created by the logic translator. We use XSB [102], as we need a Prolog system which supports the well-founded semantics. 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 Compiler with the DL Parser and the Logic Translator. Only a small number of code was enough to construct this communication module.

XSB is also used to evaluate the answer to the user's queries. The queries are parsed by the DL 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. This is the answer that the system returns to the user.

6.6 The Semantic & Syntactic Validator

This module is an embedded version of VRP [101], a parser for validating RDF documents. The RDF documents are retrieved from the Web and are checked by this module before being imported into the system. Among others, the tests that are being performed are: class hierarchy loops, property hierarchy loops, domain/range of subproperties, source/target resources of properties and types of resources.

6.7 The RDF Translator

The role of the RDF translator is to transform the RDF statements into logical facts, and the RDFS statements into logical facts and rules. This transformation allows the RDF/S information to be processed by the rules (in the form of defeasible theories) provided by the user.

For RDF data, the SWI-Prolog [91] RDF parser is used to transform them 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 "comment" elements from the RDF files.

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).

Let us consider rule *b* that captures the meaning of the subclass relation of RDFS. A class *Sc* is subclass of a class *C* when all instances of *Sc* are also instances of *C*. Stated another way, if *X* is an instance of *Sc* then it is also instance of *C*. That is exactly what rule *b* says. Rule *c* says that if *Sp* is a subproperty of P, and *X*, *Y* are respectively the subject, object of *Sp*, then *X* is also a subject of *P*, and *Y* is an object of *P*. *d* is used to declare the relation between a property and its domain: the subject of a property *P* must belong to the class which is specified by the domain *D* of the property, and *e* is used to declare the relation between a property and its range: the object of a property *P* must belong to the class which is specified by the range *R* of the property. All the above rules are created at compile-time, i.e. before the actual querying takes place. Therefore, the above rules 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.

7 A Concrete Example

In this chapter we describe a concrete example, in order to show the way that our system works and interacts with the user's requests. The reader will be able to understand better the role of each of the tools, described in Chapter 5, and perceive the data flow from the point that new data are imported into the system until the results are returned to the user.

7.1 The Scenario

Teo is a new student in the Computer Science Department at the University of Crete, in Iraklio. His brother, Adam has already been in Iraklio for two years, working for a company in the centre of the town. Teo and Adam want to rent an apartment to stay together. They wish to stay at the centre of the town, and require the apartment to have central heating. They prefer the apartment to have three rooms at least, but they are also willing to accept apartments with two rooms, on the condition that they are not on the ground floor. They are not willing to pay more than 400 Euros per month, except if the apartment has air-conditioning. In the latter case, they are willing to pay up to 450 Euros per month.

7.2 Formalization of Requirements

We show how the two brothers' requirements are expressed in defeasible logic. The predicate *acceptable(X)* is used to denote that an apartment is acceptable. The first rule says that, a priori, all apartments are acceptable.

r_1: apartment(X) $\Rightarrow$ acceptable(X)

However, any apartment not satisfying one of the required features is unacceptable. The following rules describe exceptions to the first, more general rule. Note that the exception rules are declared to be stronger than the first general rule.

r_2: $\neg$location(X, centre) $\Rightarrow$ $\neg$acceptable(X)

r_3: $\neg$heating(X, central) $\Rightarrow$ $\neg$acceptable(X)

r_4: rooms(X, Z), $Z < 2$ $\Rightarrow$ $\neg$acceptable(X)

r_5: rooms(X, Z), $Z < 3$, floor(X, ground) $\Rightarrow$ $\neg$acceptable(X)

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

Next we must represent the price the two brothers are willing to pay at most. The predicate *offer(X,Y)* denotes that Teo and Adam are willing to pay at most *Y* Euros for apartment *X*.

conflict :: offer(X,400), offer(X,450)

r_6: ⇒ offer(X,400)

r_7: aircon(X,true) ⇒ offer(X,450)

$r_7 > r_6$

An apartment is unacceptable if its price is higher than what the two brothers are willing to pay. This rule is also an exception to the general rule r_1.

r_8: price(X,Y), offer(X,Z), Y>Z ⇒ ¬acceptable(X)

$r_9 > r_1$

The next rules state that the two brothers prefer apartments with 3 rooms (even if they are on ground floor) than apartments with two rooms that are not on the ground floor. The predicate chose(X) is used to denote that an apartment meets the two brothers' requirements and preferences.

conflict :: prefer1(X), prefer2(X)

r_{10} : rooms(X, Z), Z > 2 ⇒ prefer1(X)

r_{11} : rooms(X, Z), Z = 2, ¬floor(X, ground) ⇒ prefer2(X)

r_{12} : acceptable(X), prefer1(X) ⇒ chose(X)

r_{13} : acceptable(X), prefer2(X) ⇒ chose(X)

$r_{10} > r_{11}$

7.3 Representation of Offered Apartments

In Table 7.1 we show ten different apartment offerings maintained by the broker, in table form.

Table 7.1: Offerings for ten different apartments in table form.

Apartment	Location	Heating	Rooms	Floor	Aircon	Price
a1	Mastabas	central	4	first	yes	400
a2	Kaminia	no	2	second	no	280
a3	centre	no	3	first	no	340
a4	Fortetsa	no	3	ground	no	300
a5	centre	central	2	third	no	350
a6	centre	central	3	second	yes	440
a7	centre	no	3	second	no	350
a8	Kaminia	central	4	first	no	480
a9	centre	central	3	third	no	380
a10	centre	central	3	first	no	430

The same offerings are contained as RDF data in RDF documents. An apartment offering example in RDF form is presented in Figure 7.1. The RDF Schema describing the apartments' ontology of our example (presented in Figure 7.2), is available at http://www.csd.uoc.gr/~bikakis/apartment.rdfs.

```
<?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#"
    xmlns:ap ="http://www.csd.uoc.gr/~bikakis/apartment.rdfs#">
<rdf:Description rdf:about="a1">
      <rdf:type rdf:resource="ap:apartment"/>
      <ap:location rdf:datatype="&xsd;istring">Mastabas</ap:location>
      <ap:heating rdf:datatype="&xsd;string">central</ap:heating>
      <ap:rooms rdf:datatype="&xsd;integer">4</ap:rooms>
      <ap:floor rdf:datatype="&xsd;string">first</ap:floor>
      <ap:aircon rdf:datatype="&xsd;boolean">true</ap:aircon>
      <ap:price rdf:datatype="&xsd;integer">400</ap:price>
</rdf:Description>
</rdf:RDF>
```

Figure 7.1: An apartment offering example in RDF form.

```
<?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="house"></rdfs:Class>
<rdfs:Class rdf:ID="apartment">
      <rdfs:subClassOf rdf:resource="#house"/></rdfs:Class>
<rdfs:Class rdf:ID="cottage">
      <rdfs:subClassOf rdf:resource="#house"/></rdfs:Class>
<rdf:Property rdf:ID="houseID">
      <rdfs:domain rdf:resource="#house"/>
      <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="location">
      <rdfs:domain rdf:resource="#house"/>
      <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="heating">
      <rdfs:domain rdf:resource="#house"/>
      <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="rooms">
      <rdfs:domain rdf:resource="#house"/>
      <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>
<rdf:Property rdf:ID="aircon">
      <rdfs:domain rdf:resource="#house"/>
      <rdfs:range rdf:resource="&xsd;boolean"/>
</rdf:Property>
<rdf:Property rdf:ID="price">
      <rdfs:domain rdf:resource="#house"/>
      <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>
<rdf:Property rdf:ID="floor">
      <rdfs:domain rdf:resource="#apartment"/>
      <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="garden">
      <rdfs:domain rdf:resource="#cottage"/>
      <rdfs:range rdf:resource="&xsd;boolean"/>
</rdf:Property>
   </rdf:RDF>
```

Figure 7.2: The RDF Schema of the apartments' ontology.

The facts concerning the apartment offering of "*a1*", which are created by the RDF translator have the following form:

apartment(a1).

location(a1,Mastabas).

heating(a1,central).

rooms(a1,4).

floor(a1,first).

aircon(a1,true).

price(a1,400).

The rules, described in 7.2 are translated into the corresponding logic program by the Logic Translator, and along with the facts representing the apartment offerings are compiled by the Logic Compiler.

7.4 Selecting the Appropriate Apartments

The role of the Evaluator is to find the apartments that meet the two brothers' requirements and preferences. Based on the rules, described in 7.2 and the facts representing the apartments offerings, described in 7.3:

- Apartment a1 is unacceptable because it is not located in the centre of the town. (rule r_2).
- Apartment a2 is unacceptable because it is not located in the centre of the town (rule r_2), and it does not have central heating (rule r_3).
- Apartment a3 is unacceptable because it does not have central heating (rule r_3).
- Apartment a4 is unacceptable because it is not located in the centre of the town (rule r_2), and it does not have central heating (rule r_3).
- Apartment a7 is unacceptable because it does not have central heating (rule r_3).
- Apartment a8 is unacceptable because it is not located in the centre of the town (rule r_2).
- Apartment a10 is unacceptable because its price (430) is higher than what Teo and Adam are willing to pay (400; rule r_8).
- Apartments a5, a6, and a7 are acceptable (rule r_1).

Between the acceptable apartments, the system "decides" that a6 and a7 are those that suit the two brothers' preferences (rules r_{10}, r_{11}).

8 Performance Evaluation

In this chapter we present the experimental evaluation we made in order to measure the performance of our system, and to compare it with the performance of other defeasible reasoning systems, namely *Deimos* [64], and *d-Prolog* [28]. Firstly, we describe the test defeasible theories that we use for the experiments, and then we present and try to interpret the experimental results.

8.1 Design of the experiments

We employed the *DTScale* tool of Deimos to create the experimental tests for the evaluation. The tests are in fact defeasible theories, consisting of a large number of facts, rules (strict and defeasible), and superiority relations. The test theories do not include defeaters, as they are not supported by our system. They do not also contain user-defined conflicting literals, as this feature is not supported by the other systems. In the experiments we focus on defeasibly inference. Below we present the form of the theories that we created:

8.1.1 Chain Theories

Chain theories **chain**(n) 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.

$$\textbf{chain}(n) = \begin{cases} a_o \\ r_1 : a_o \Rightarrow a_1 \\ r_2 : a_1 \Rightarrow a_2 \\ \vdots \\ r_n : a_{n-1} \Rightarrow a_n \end{cases}$$

8.1.2 Circle Theories

Circle theories **circle**(n) consist of n defeasible rules $a_i \Rightarrow a_{(i+1) \bmod n}$.

$$\textbf{circle}(n) = \begin{cases} r_0 : a_o \Rightarrow a_1 \\ r_1 : a_1 \Rightarrow a_2 \\ \vdots \\ r_{n-1} : a_{n-1} \Rightarrow a_0 \end{cases}$$

Any proof $+\partial a_i$ will fail. A variant **circles**(n) uses only strict rules.

8.1.3 Levels Theories

Levels theories **levels**(n) 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.

$$\textbf{levels}(n) = \begin{cases} \\ r_0 : \Rightarrow a_0 \\ \underline{r_1 : a_1 \Rightarrow \neg a_0} \\ r_2 : \Rightarrow a_1 \\ r_3 : a_2 \Rightarrow \neg a_1 \\ \underline{r_3 > r_2} \\ r_4 : \Rightarrow a_2 \\ \underline{r_5 : a_3 \Rightarrow \neg a_2} \\ \vdots \\ r_{4n+2} : \Rightarrow a_{2n+1} \\ r_{4n+3} : a_{2n+2} \Rightarrow \neg a_{2n+1} \\ \underline{r_{4n+3} > r_{4n+2}} \\ \underline{r_{4n+4} : \Rightarrow a_{2n+2}} \end{cases}$$

8.1.4 Teams theories

Teams theories **teams**(n) 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.

$$\textbf{teams}(n) = \textbf{block}(a_o, n)$$

where, if p is a literal, and $r_1,\ldots,r_4$ are unique labels:

$$\textbf{block}(p,0) = \begin{cases} r_1 : \Rightarrow p \\ r_2 : \Rightarrow p \\ r_3 : \Rightarrow \neg p \\ r_4 : \Rightarrow \neg p \\ r_1 > r_3 \\ r_2 > r_4 \end{cases}$$

and, if $n > 0$, $a_1,\ldots,a_4$ are new unique literals, and $r_1,\ldots,r_4$ are new unique labels:

$$\textbf{block}(p,n) = \begin{cases} r_1 : a_1 \Rightarrow p \\ r_2 : a_2 \Rightarrow p \\ r_3 : a_3 \Rightarrow \neg p \\ r_4 : a_4 \Rightarrow \neg p \\ r_1 > r_3 \\ r_2 > r_4 \\ block(a_1, n-1) \\ block(a_2, n-1) \\ block(a_3, n-1) \\ block(a_4, n-1) \end{cases}$$

8.1.5 Tree theories

In tree theories **tree**(n,k) a_0 is at the root of a k-branching tree of depth n in which every literal occurs once.

$$\textbf{tree}(n,k) = \textbf{block}(a_0, n, k)$$

where, if p is a literal, $n > 0$, r is a new unique label, and $a_1,a_2,\ldots,a_k$ are new unique literals:

$$\textbf{block}(p, n, k) = \begin{cases} r : a_1, a_2, \ldots, a_k \Rightarrow p \\ block(a_1, n-1, k) \\ block(a_2, n-1, k) \\ \vdots \\ block(a_k, n-1, k) \end{cases}$$

and **block**(p, 0, k) = {p.

8.1.6 Directed Acyclic Graph Theories

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

$$\mathbf{dag}(n, k) = \begin{cases} a_{kn+1} \\ a_{kn+2} \\ \vdots \\ a_{kn+k} \\ r_0 : a_1, a_2, \ldots, a_k \Rightarrow a_0 \\ r_1 : a_2, a_3, \ldots, a_{k+1} \Rightarrow a_1 \\ \vdots \\ r_{nk} : a_{nk+1}, a_{nk+2}, \ldots, a_{nk+k} \Rightarrow a_{nk} \end{cases}$$

8.1.7 Theory sizes

In Table 8.1, 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.

Table 8.1: Size of the test theories

theory	*facts*	*rules*	*priorities*	*size*
$\mathbf{chain}(n)$	1	n	0	$2n+1$
$\mathbf{chain}^{\mathbf{s}}(n)$	1	n	0	$2n+1$
$\mathbf{circle}(n)$	0	n	0	$2n$
$\mathbf{circle}^{\mathbf{s}}(n)$	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	$2m + 4mn + 4mnk$

8.2 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. We used a paging file of 1440MB, so as to have enough virtual memory for the experiments.

For the experimental evaluation of our system, we use 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 use 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.

We compile Deimos using the Glasgow Haskell Compiler 6.2.1 for Windows. The execution times are measured using the –m option of DTScale (this is the tool of Deimos used for the creation of the test theories). We also use 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 use the 5.2.13 version of SWI-Prolog for Windows. The times presented in the experiments are those measured by 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.

8.3 Experimental Results

The tables describe 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. So, we could not test our system and d-Prolog in cases of theories with size bigger than 20000.

In the tables below, ∞ 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 8.2 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 8.3, we record the times in the case of theories with disputed references, namely *levels(n), levels-(n)* and *teams*.

Table 8.2: Execution times for theories with Undisputed Inferences

	Problem Size	**Our System**	**Deimos**	**d-Prolog**
chains(n)				
n = 1000	2001	0.02	0.16	0.00
n = 2000	4001	0.03	0.60	0.01
n = 5000	10001	0.06	3.60	0.02
chain(n)				
n = 1000	2001	0.05	0.41	0.13
n = 2000	4001	0.08	1.44	0.25
n = 5000	10001	0.10	8.59	0.62
circles(n)				
n = 1000	2000	0.03	0.16	∞
n = 2000	4000	0.04	0.61	∞
n = 5000	10000	0.07	3.67	∞
circle(n)				
n = 1000	2000	0.07	0.25	∞
n = 2000	4000	0.10	0.90	∞
n = 5000	10000	0.13	5.60	∞
tree(n,k)				
n = 6, k = 3	2185	0.02	0.22	0.06
n = 7, k = 3	6559	0.08	1.37	0.15
n = 8, k = 3	19681	0.22	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	0.03	0.06	$*$
n = 100, k = 10	11021	0.11	0.49	$*$

Table 8.3: Execution times for theories with Disputed Inferences

	Problem Size	Our System	Deimos	d-Prolog
levels-(n)				
n = 10	67	0.02	0.00	1.61
n = 20	127	0.02	0.01	-
n = 100	607	0.06	0.06	-
n = 1000	6007	0.12	3.53	-
levels(n)				
n = 10	78	0.02	0.00	1.70
n = 20	148	0.02	0.01	-
n = 100	708	0.06	0.06	-
n = 1000	7008	0.39	3.78	-
teams(n)				
n = 3	594	0.11	0.05	-
n = 4	2386	0.89	0.26	-
n = 5	9554	2.30	1.15	-

As it is obvious from the two tables, the cases that our system performs best is the theories that contain undisputed inferences, or the theories with disputed inferences but with not many superiority relations. For example, if we compare the time that our system spends on the *tree(8,3)* theory, which contains only facts and rules with undisputed inferences, with the time that it spends on *teams(5)* which contains a large number of conflicting rules and superiority relations between them, we can see that although the *tree* theory is double in size, the execution time for this theory is 10 times smaller. The reason for this is that theories with disputed inferences result in Prolog files with many more program clauses, as we commented in Chapter 5.

Comparing to Deimos our system performs better in most of the cases. The only exception is the *teams* theories, which contain many conflicting rules and superiority relations. In general, when the size of the theories is relatively small, the two systems have more or less the same performance. But in the case of large theories, our system performs obviously much better. However, we should mention that the time that Deimos spends in order to build and compile the same theories (which is not recorded in the tables above) is by far less than the corresponding time of our system. Moreover, in the case of theories with very large sets of rules and priorities (size > 20000), our system is unable to compile the logic files, as XSB runs out of memory.

Comparing to d-Prolog, our system performs better in all the cases that there are defeasible rules in the test theories. 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 (for example in the case of the *chains* theories), due to the direct execution of such rules. However, d-Prolog shows its incompleteness, comparing to the other two systems, when it loops on *circle(n)*, which are cyclic theories.

9 Conclusions and Future Work

9.1 General Conclusions

In this report, we describe our work on the implementation of a defeasible reasoning system for the web. At first, we reason why such a system can be useful in the layered development of the Semantic Web [13]. Then, we describe the logic (Defeasible Logic [5][73]), on which the system is based on, presenting its declarative capabilities, and its low computational complexity. We present in detail the architecture of the system, its rationale, and its functionality, and compare its performance capabilities with other similar defeasible reasoning systems.

Defeasible Logic seems to be a very useful tool for reasoning about resources of the Web, due to its representational capabilities and its low computational complexity. The translation of defeasible theories into logic programs, that we present in Chapter 5, is an effort to efficiently combine this logic with the resources that are expected to be available in the Semantic Web. The implementation of tools that translate data contained in RDF documents into logical rules and facts, and represent defeasible theories in an XML format, as those described in Chapter 6, moves in the same direction: in the integration of rule systems with the layers of the Semantic Web that have been so far implemented. Although the use of nonmonotonic rule systems still remains an issue between the developers of the Semantic Web, it seems that it will not take much longer, until they find their role in the development of the Semantic Web.

The system described in this report, moves one step forward from the other existing defeasible reasoning systems; it is not an isolated system, as d-Prolog [28] or Deimos [64]. It has been designed to integrate with the Semantic Web, as it has the ability to treat RDF data and RDF ontologies, and to use an XML-based format to express the defeasible theories. This makes it suitable for integrated web applications for brokering, bargaining, automated agent negotiation, and personalization. Moreover, it is more flexible and adaptable to different intuitions within defeasible reasoning, as the ambiguity blocking and ambiguity propagating behavior, and the use of conflicting literals. Finally, its implementation is declarative, because it interprets the "not" operator using Well-Founded Semantics. Its main drawbacks are two: it cannot handle very large sets of rules; and it cannot compute all answers as it implements a backward-chaining theorem prover. The first problem is caused by the very large logic programs that result from the translation of defeasible theories with many conflicting rules, and by the restricted capabilities of the logic programming system (XSB [102]) that we use as the Logic Compiler. The second is probably caused by the method that we use to translate defeasible theories into logic programs, which does not permit the Evaluator to calculate all answers, given a query containing a free variable. The solution of these problems is part of our planned future work, which is described in the next section.

9.2 Planned Future Work

Our planned future work includes:

- Adding arithmetic capabilities to the rule language, and using appropriate constraint solvers in conjunction with logic programs. This will allow us to add more expressional capabilities to the rule language, and make it more suitable for expressing business rules and contracts.
- Add two kinds of negation into the object language; both classical negation and negation as failure.
- Implementing load/upload functionality in conjunction with an RDF repository, such as RDF Suite [1] and Sesame [24]. This additional functionality will promote the integration of the system with the Semantic Web, and will make it more suitable for building web-based applications.
- 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 [43].
- Studying in the translation of defeasible theories into logical programs, through the use of a meta-program, as that proposed in [7]. A such translation can reduce the size of the logical programs, improving the performance of the system.
- Applications of defeasible reasoning and the developed implementation for brokering, bargaining, automated agent negotiation, and personalization.

10 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]. J. Alferes and L. Pereira. Updates plus Preferences. In M. Aciego, I. de Guzman. G. Brewka, and L. Pereira, editors, *Proc. 7th European Workshop on Logics in Artificial Intelligence*, volume 1919 of *Lecture Notes in Computer Science*, p. 345-360. Springer, 2000.

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

[4]. G. Antoniou, D. Billington, G. Governatori and M.J. Maher. Defeasible logic versus logic programming without negation as failure. *Journal of Logic Programming,* 42(1):47-57, 2000. Grosof, B.N. Prioritized conflict handling for logic programs. In *Proc. Int. Logic Programming Symposium*, J. Maluszynski, Ed. MIT Press, 197-211, 1997.

[5]. G. Antoniou, D. Billington, G. Governatori and M.J. Maher. Representation Results for Defeasible Logic. *ACM Transaction on Computational Logic*, 2, 255-287, 2001.

[6]. 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.

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

[8]. D.E. Appelt and K. Konolige. A Nonmonotonic Logic Reasoning about Speech Acts and Belief Revision. In Reinfrank et. al. eds, *Nonmonotonic Reasoning, Proc. 2nd International Workshop*, Springer LNAI 346.

[9]. 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, California, 2004.

[10]. F. Baader, D. Calvanese, D. McGuiness, D. Nardi, and P. Patel-Schneider (eds.). *The Description Logic Handbook*. Cambridge University Press, 2002.

[11]. C. Baral, M.Gelfond. Logic Programming and knowledge representation. *Journal of Logic Programming,* 19,20, 73-148, 1994.

[12]. T. Berners-Lee, R. Fielding, and L. Masinter (eds.). IETF (Internet Engineering Task Force) RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax, 1998.

[13]. T. Berners-Lee, J. Hendler, O. Lassila. "The Semantic Web". *Scientific American* 284(5), p. 34-43, 2001.

[14]. D. Billington. Defeasible logic is stable. *Journal of Logic and Computation,* 3(4):379-400, 1993.

[15]. A. Bondarenko, P.M. Dung, R.A. Kowalski & F. Toni, An abstract argumentation-theoretic approach to default reasoning. *Artificial Intelligence* 93:63-101, 1997.

[16]. T. Bray, D, Hollander, and A. Layman (eds.). Namespaces in XML, 1999. http://www.w3.org/TR/REC-xml-names.

[17]. T. Bray, J. Paoli, C.M. Sperberg-McQueen, and E. Maler (eds.). Extensive Markup Language (XML) 1.0 (Second Edition), 2000. http://www.w3.org/TR/REC-xml.

[18]. G. Brewka, Reasoning About Priorities in Default Logic, *Proc. AAAI-94*, Seattle, 1994.

[19]. G. Brewka. Well-founded semantics for extended logic programs with dynamic preferences. *Journal of Artificial Intelligence Research* 4:19-36. 20, 1996.

[20]. G. Brewka. On the Relationship between Defeasible Logic and Well-Founded Semantics. In *Proc. Logic Programming and Nonmonotonic Reasoning Conference*, LNCS 2173, 121–132, 2001.

[21]. G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109(1-2):297-356, 1999.

[22]. G. Brewka and T. Eiter. Prioritizing default logic. In St. Holldobler, editor, *Intellectics and Computational Logic-Papers in Honour of Wolfgang BibelI,* p. 27-45. Kluwer Academic Publishers, 2000.

[23]. D. Brickley and R.V. Guha (eds.). RDF Vocabulary Description Language 1.0: RDF Schema, 2003. http://www.w3.org/TR/rdf-schema.

[24]. J. Broekstra, A. Kampman and F. van Harmelen 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, 2003.

[25]. M. Cadoli, M. Schaerf. A Survey on Complexity Results for Non-monotonic Logics. *Journal of Logic Programming* , Vol. 17, 127-160, 1993.

[26]. K.L. Clark. Negation as failure. In *Logic and Databases*, eds. H. Gallaire and J. Minker, 292-322, Plenum, New York, 1978.

[27]. A. Colmerauer, H. Kanoui, R.Pasero and P. Roussel. Un système de communication homme-machine en français. Technical report, Groupe de Intelligence Artificielle Université de Aix-Marseille II, Marseille, 1973.

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

[29]. B. Cui and T. Swift. Preference logic grammars: Fixed-point semantics and application to data standardization. *Artificial Intelligence*, 138(1-2):117-147, 2001.

[30]. S. Decker, Dan Brickley, Janne Saarela, and Jurgen Angele. A query and inference engine for RDF. In *QL'98 – The Query Languages Workshop,* Boston, USA, 1998.

[31]. J. Delgrande, T. Schaub, H. Tompits, K. Wang. A Classification and Survey of Preference Handling Approaches in Nonmonotonic Reasoning. *Computational Intelligence*, 20(12):308-334, 2004.

[32]. J. Delgrande and T. Schaub. Expressing preferences in default logic. *Artificial Intelligence,* 123(1-2):41-87, 2000.

[33]. J. Doyle. A truth maintenance system *Artificial Intelligence,* 12:231-272, 1979.

[34]. P.M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming, and *n*-person games. *Artificial Intelligence* 77:321-357, 1995.

[35]. D. W. Etherington. Formalizing Nonmonotonic Reasoning Systems, *Artificial Intelligence* 31: 41-85, 1987.

[36]. M. Gelfond, V. Lifschitz. Logic programs with classical negation. In *Proc. 7th International Conference of Logic Programming,* 1990.

[37]. M. Gelfond, V. Lifschitz, H. Przymusinska and M. Truszczynski. Disjunctive defaults. In J. Allen, R. Fikes, and E. Sandewall, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference*, pages 230-237, San Mateo, California, April 22-25, 1991.

[38]. T. Gordon. *The Pleading Game: An Artificial Intelligence Model of Procedural Justice.* Dissertation, Technische Hochschule Darmstadt, Germany, 1993.

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

[40]. G. Governatori, M.J. Maher, G. Antoniou & D. Billington, Argumentation Semantics for Defeasible Logics, *Proc. Pacific Rim Conf. on Artificial Intelligence*, LNAI 1886, 27-37, 2000.

[41]. B. N. Grosof. Business rules for electronic commerce. http://www.research.ibm.com/rules/papers.html, 1999. IBM Research.

[42]. B. N. Grosof. Prioritized conflict handling for logic programs. In *Proc. Int. Logic Programming Symposium*, J. Maluszynski, Ed. MIT Press, 197-211

[43]. 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 (WWW-2003)*, ACM Press, 2003.

[44]. 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.

[45]. C.E. Hewitt. PLANNER : A Language for proving theorems in robots. *First International Joint Conference on Artificial Intelligence,* pages 295-301, 1969.

[46]. I. Horrocks. The FaCT System, http://www.cs.man.ac.uk/~horrocks/FaCT/.

[47]. I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, W3C Member Submission, http://www.w3.org/Submission/2004/SWRL, 2004.

[48]. J.F. Horty, R.H. Thomasson & D.S. Touretzky, A sceptical theory of inheritance in nonmonotonic semantic networks. *Artificial Intelligence* 42:311-348, 1990.

[49]. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM,* 42:741-843, July 1995.

[50]. K. Konolige. On the relation between default and autoepistemic logic. *Artificial Intelligence,* 35(3):343-382, July 1988.

[51]. K. Konolige. Quantifying in Autoepistemic Logic. *Fundamental Informaticae* 15(3-4), 1991.

[52]. K. Konolige. Hierarchic Autoepistemic Theories for Nonmonotonic Reasoning: Preliminary Report. In Renfrank et. al. eds, *Nonmonotonic Reasoning, Proceedings 2nd International Workhop,* Springer LNAI 346.

[53]. S. Kraus, D. Lehmann and M. Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence* 44: 167-207.

[54]. O. Lassila and R. Swick (eds.). Resource Description Framework (RDF) Model and Syntax Specification 1, 1999. http://www.w3.org/TR/REC-rdf-syntax.

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

[56]. Lex & Yacc, http://dinosaur.compilertools.net

[57]. 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.

[58]. V. Lifschitz. Closed-world databases and circumscription. *Artificial Intelligence*, 27:229-235, 1985.

[59]. F. Lin & Y. Shoham, Argument systems. A uniform basis for nonmonotonic reasoning. *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, 245-255. San Mateo, CA: Morgan Kaufmann Publishers Inc, 1989.

[60]. R.P. Loui, Defeat among arguments: a system of defeasible inference. *Computational Intelligence* 2:100-106, 1987.

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

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

[63]. M. Maher, A Model-Theoretic Semantics for Defeasible Logic, *Proc. Workshop on Paraconsistent Computational Logic*, 67 - 80, 2002.

[64]. 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: 483-501, 2001

[65]. D. Makinson. General patterns in nonmonotonic reasoning. In *Handbook of Logic in Artificial Intelligence and Logic Programming* Vol. 3, Oxford University Press, 35-110.

[66]. W. Marek, M. Truszczynski, Relating Autoepistemic and Default Logics, in R. Brachman, H. Levesque and R.Reiter eds, *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning,* Toronto, Ontario, Morgan Kaufmann, 276-288.

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

[68]. J. McCarthy. Programs with common sense. *Mechanization of Thought Processes, Proceedings of the Symposium of the National Physics Laboratory,* vol. 1, 77-84. London. Reprinted in *Semantic Information Processing*, pages 403-418, The MIT Press, Cambridge, MA, 1968, 1958.

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

[70]. J. McCarthy and P.J. Hayes. Some philosophical problems form the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463-502. Edinburgh University Press, 1969.

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

[72]. D. Nute. Defeasible Reasoning and Decision Support Systems. *Decision Support Systems* 4, 97-110, 1988.

[73]. D. Nute. Defeasible logic. In *Handbook of Logic in Artificial Intelligence and Logic Programming, Vol. 3*, p. 353-395. Oxford University Press, 1994.

[74]. P.F. Patel-Schneider and D. Fensel. Layering the Semantic Web: Problems and Directions. In I. Horrocks and J. Hendler, editor, *The Semantic Web – ISWC 2002*, LNCS 2342, p. 16-29, Springer-Verlag, 2002.

[75]. P.F. Patel-Schneider, P. Hayes, and I. Horrocks (eds.). OWL Web Ontology Language Semantics and Abstract Syntax, 2003. http://www.w3.org/TR/owl-semantics.

[76]. J.L. Pollock, Defeasible reasoning, *Cognitive Science* 11:481-518, 1987.

[77]. D. Poole, On the Comparison of Theories: Preferring the Most Specific Explanation, *Proc. IJCAI-85*, Los Angeles, 1985

[78]. H. Prakken, Logical Tools for Modelling Legal Argument, dissertation, VU Amsterdam, 1993.

[79]. H. Prakken. *Logical Tools for Modelling Legal Argument. A Study of Defeasible Reasoning in Law.* Dordrecht etc.: Klwer Law and Philosophy Library, 1997.

[80]. H. Prakken & G. Sartor, A dialectical model asserting conflicting arguments in legal reasoning. *Artificial Intelligence and Law* 4:331-368, 1996.

[81]. H. Prakken & G. Sartor, Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-classical Logics* 7:25-75, 1997.

[82]. T. Przymusinski. Stable semantics for disjunctive programs. *New Generation Computing, 9,* 401-424, 1991.

[83]. R. Reiter. On Closed World Data Bases. In *Logic and Bases*, eds. H. Gallaire and J.Minker, 55-76, Plemum, New York, 1978.

[84]. R. Reiter. On reasoning by default. In *Proceedings of TINLAP-2, Theoretical Issues in Natural Language Processing-2,* pages 210-218, University of Illinois at Urbana-Champaign, 1978.

[85]. R. Reiter, A Logic for Default Reasoning, *Artificial Intelligence* 13, 81-132, 1980.

[86]. R. Reiter and G. Criscuolo. On interacting defaults. In *Proceedings of INJCAI-81*, pages 270-276, 1981.

[87]. R. Reiter, J. De Kleer, Foundations of Assumption based Truth Maintenance Systems: Preliminary Report, in *Proceedings of the Conference of the American Association of Artificial Intelligence,* Seattle, WA, 1987.

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

[89]. E. Sandewall. An approach to the frame problem and its implementation. In *Machine Intelligence 7*, pages 195-204. Edinburgh University Press, 1985.

[90]. M. Sintek, S. Decker. TRIPLE – A Query, Inference, and Transformation Language for the Semantic Web. In *Proceedings of the First International Semantic Web Conference*, Sardinia, 2002.

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

[92]. A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics, 5,* 285-309, 1955.

[93]. S. Thompson, D. Beech, M. Maloney, and N. Mendelson (eds.). XML Schema Part 1: Structures, 2001. http://www.w3.org/TR/xmlschema-1.

[94]. D. Touretzky, The Mathematics of Inheritance, Pitman Research Notes in Artificial Intelligence, London, 1986.

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

[96]. D. Touretzky, Thomason, Richmond H., Horty, John F., A Skeptic's Menagerie: Conflictors, Preemptors, Reinstaters, and Zombies in Nonmonotonic Inheritance, *Proc. 12th IJCAI*, Sydney, 1991

[97]. M. Truszczynski, Embedding Default Logic into Modal Nonmonotonic Logics, in A. Nerode, W. Marek and V. S. Subrahmanian eds, *Logic Programming and Non-monotonic Reasoning, Proceedings of the 1st International Workshop,* MIT Press, Cambridge, Mass., 151-165, 1991.

[98]. N. Vassiliades, G. Antoniou and Y. Vlahavas. DR-DEVICE: A Defeasible Reasoning Systems for the Web. Submitted, 2004.

[99]. G.A. W. Vreeswijk, *Studies in Defeasible Argumentation.* Doctoral dissertation, Department of Computer Science, Free University of Amsterdam, 1993.

[100]. G.A. W. Vreeswijk, Abstract argumentation systems. *Artificial Intelligence* 90:225-279, 1997.

[101]. VRP. The ICS-FORTH Validating Rdf Parser –VRP (2004). Available at: http://139.91.183.30:9090/RDF/VRP/

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

[103]. Y. Zhang and N. Foo. Answer sets for prioritized logic programs. In J. Maluszynski, editor, *Proceedings of the International Symposium on Logic Programming*, p. 69-84. MIT Press, 1997.