

On the problem of computing Ambiguity Propagation and Well-Founded Semantics in Defeasible Logic

Ho-Pun Lam^{1,2} and Guido Governatori²

¹ School of Information Technology and Electrical Engineering
The University of Queensland, Brisbane, Australia

² NICTA*, Queensland Research Laboratory, Brisbane, Australia

Abstract. In this paper we present the well founded variants of ambiguity blocking and ambiguity propagating defeasible logics. We also show how to extend SPINdle, a state of the art, defeasible logic implementation to handle all such variants of defeasible logic.

Keywords: Ambiguity Propagation, Well-Founded Semantics, Defeasible logics, Consequences finding

1 Introduction

Defeasible Logic (DL) [1,2] is a skeptical approach to non-monotonic reasoning. It is based on a logic programming-like language and is a simple, efficient but flexible formalism capable of dealing with many different intuitions of non-monotonic reasoning in a natural and meaningful way [3].

The main advantage of using DL over other non-monotonic formalisms is certainly due to its low computation complexity: Conclusions of DL can be derived in linear time (wrt the size of a theory) [4] and several efficient implementations exist [5,6,7]. Besides, due to its built-in preference handling facilities, it also capable to derive plausible conclusions from incomplete and conflicting information in a declarative way.

Recently, [8] has investigated the relationships among several variants of DL, capturing the intuitions of different reasoning issues, such as ambiguity blocking and propagation, and team defeat. Our focus is on the computational aspect of these variants. We have devised algorithms to compute in linear time the extensions of the ambiguity propagation variants and well-founded semantics of DL. For the well-founded variant we have established a way to compute the unfounded set of a defeasible theory. In addition, by combining the algorithms together, we can handle the well-founded variants of ambiguity blocking and ambiguity propagation of DL maintaining linear complexity.

The outline of this paper is as follows. Section 2 gives a brief introduction and modular construction to the syntax and semantics of defeasible logic. Section 3 and 4 describe the algorithms proposed to compute the ambiguity propagation variant and well-founded semantics of DL respectively, followed by a conclusion.

* NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

2 Basics of Defeasible Logic

In this Section we provide a short outline of DL and the construction of variants capturing different intuitions of non-monotonic reasoning based on a modular and parametrised definition of the proof theory of the logic. For the full details, we refer to [9,10,8].

A defeasible theory D is a triple $(F, R, >)$ where F and R are finite set of facts and rules respectively, and $>$ is an acyclic superiority relation on R . *Facts* are logical statements describing indisputable facts; they are represented by (atomic) propositions (i.e., literals). A rule r describes the relations between a set of literals (the *antecedent* $A(r)$, which can be empty) and a literal (the *consequent* $C(r)$). In writing rules we omit set notation for antecedents. There are three types of rules: *strict* rules ($r : A(r) \rightarrow C(r)$), *defeasible* rules ($r : A(r) \Rightarrow C(r)$), and *defeaters* ($r : A(r) \rightsquigarrow C(r)$). Strict rules are rules in the classical sense, the conclusion follows every time the antecedents hold; a defeasible rule is allowed to assert its conclusion in case there is not contrary evidence to the conclusion. *Defeaters* cannot support conclusions but they provide contrary evidence to them. The *superiority relation* $>$ describes the relative strength of rules, and it is used to obtain a conclusion when there are applicable conflicting rules.

DL is able to distinguish positive conclusions from negative conclusions, that is literals that can be proved and literals that are refuted, in addition it is able to determine the strength of a conclusion, i.e., whether something is concluded using only strict rules and facts or whether we have a defeasible conclusion, a conclusion can be retracted if more evidence is provided. Accordingly, for a literal p we can have the following four types of conclusions, called tagged literals: $+\Delta p$ (p is definitely provable), $-\Delta p$ (p is definitely refuted), $+\partial p$ (p is defeasible provable), and $-\partial p$ (p is defeasibly refuted). At the heart of DL we have its proof theory that tell us how to derive tagged literals. A *proof* is a sequence of tagged literals obeying proof conditions corresponding to inference rules. The inference rules establish when we can add a literals at the end of a sequence of tagged literals based on conditions on the elements of a theory and the previous tagged literals in the sequence.

The structure of the proof conditions has an argumentation flavour:

To prove $+\partial p$

Phase 1: There is an applicable rule for p and

Phase 2: For every rule for $\sim p$ (the complement of p) either

Sub-Phase 1: the rule is discarded, or

Sub-Phase 2: the rule is defeated by a (stronger) rule for p

The notion of a rule being applicable means that all the antecedents of the rule are provable (with the appropriate strength); a rule is discarded if at least one of the antecedents is refuted (with the appropriate strength), and finally a rule is defeated, if there is a (stronger) rule for the complement of the conclusion that is applicable (again with the appropriate strength).

The above structure enables us to define several variants of DL by giving different parameters (i.e., this is what we mean ‘with the appropriate strength’ in the previous paragraph). In particular we address the distinction between ambiguity blocking and ambiguity propagation.

3 Ambiguity Propagation

Intuitively a literal p is *ambiguous* iff there exist two chains of reasoning with one supports the conclusion p is true while another supports the conclusion $\neg p$ is true, one supports the conclusion p is true whereas one supports the conclusion $\neg p$ is true, and the superiority relation does not resolve this conflict.

Example 1. Consider the following theory:

$$\begin{array}{lll} \Rightarrow a & a \Rightarrow c & \Rightarrow \neg d \\ \Rightarrow b & b \Rightarrow \neg c & \neg c \Rightarrow d \end{array}$$

Literals c and $\neg c$ are ambiguous, since in both cases we have ‘chains’ or reasoning leading to them. More specifically, we can prove both $+\partial a$ and $+\partial b$ since there are no rules for their complements, and their rules have empty antecedents, thus the condition that all element of the antecedent are provable is trivially satisfied. At this stage, we have applicable rules for c and $\neg c$. Since the superiority relation is empty we cannot solve the conflict, thus both literals are refuted, i.e., we prove $-\partial c$ and $-\partial \neg c$. Then we have the rule $\neg c \Rightarrow c$ and $\Rightarrow \neg d$. In this case the first rule is discarded since $\neg c$ is refuted. This allows us to conclude $+\partial \neg d$. In this case the ambiguity of c and $\neg c$ is restricted to them and it does not propagate to literals depending on them. we refer to this kind of reasoning as *ambiguity blocking*. On the other hand, one can reason as follows: we have a chain of rules leading to d and none of these rules is overruled by other rules; similarly we have chain of rules leading to $\neg d$, and, again the rules in this chain are not defeated. Thus we have no way to solve the conflict, thus d and $\neg d$ are ambiguous and we have to refute them; in this case we speak of *ambiguity propagation*: there are conflicts in intermediate steps of chains of rules, but these are not defeated. These two lines of reasoning are both valid and appropriate in particular applications. Accordingly, we want to be able to model both of them. The proof conditions described above are suitable for ambiguity blocking. To capture ambiguity propagation one has to make more hard to provide an argument, and easier to give a counter-argument. To this end the notion of support (Σ) is introduced.

A literal p is supported if

- Phase 1: There is a supported rule for p and
- Phase 2: Every rule for $\sim p$ stronger than it is not applicable

Armed with the notion just defined, the ambiguity propagating version of DL is obtained from the same scheme as that of the ambiguity blocking DL, where we stipulate that a rule is *discarded* if at least one of the elements in its antecedent are not supported (according the construction just given). In addition, a rule is *supported* if all the elements of its antecedent are supported.

3.1 Computing Consequences in DL with Ambiguity Propagation

Following the idea of [4] the algorithms to compute the extension of the ambiguity propagation variant of a DL are based on a series of (theory) transformations that allow

us to (1) assert whether a literal is provable or not (and its strength) and (2) progressively reduce and simplify a theory. The key ideas rely on the fact that once we have established that a literal is positively provable, we can remove it from the antecedent of rules that contain it without affecting the extension of the theory. Similarly, when it is established that a literal p cannot be proven then those rules with p in their antecedent become inapplicable and the rule can be removed from the theory.

Algorithm 1 computes the consequences of ambiguity propagation variants of DL. In the algorithm, p ranges over literals and s ranges over conclusions. \mathcal{S} holds those proven conclusions that have not been used to derived further consequences; while \mathcal{H} accumulates over the set of conclusions that have been proven and used. \mathcal{D} is the input defeasible theory without superiority relations and defeaters³.

Algorithm 1: Inference algorithm for ambiguity propagation

Algorithm: *ComputeDefeasibleAP*(\mathcal{D})
Data: $\mathcal{D} = (F, R, \emptyset)$: a defeasible theory
Result: \mathcal{H}_{ap} : set of defeasible conclusions derived

```

1 initialize  $\mathcal{S}$ 
2  $\mathcal{H}_{ap} = \emptyset$ 
3 while  $\mathcal{S}$  is not empty do
4    $\mathcal{S} = \mathcal{S} \setminus \{s\}$  for some  $s \in \mathcal{S}$ ;
    $\mathcal{H}_{ap} = \mathcal{H}_{ap} \cup \{s\}$ 
5   switch  $s$  do
6     case  $+\partial_{ap}p$ :
7       foreach  $r \in R_{sd} : p \in A(r)$  do
8         remove  $p$  from  $A(r)$ 
9         if  $A(r)$  is empty then
10           $h = C(r)$ 
11          if  $\neg h \in \Sigma^+$  then
12             $\mathcal{S} = \mathcal{S} \cup \{-\partial_{ap}h\}$ 
13            remove:
14               $\forall r \in R_{sd} : h \in A(r)$ 
15            else if  $R_{sd}[\neg h]$  is null then
16               $\mathcal{S} = \mathcal{S} \cup \{+\partial_{ap}h\}$ 
17              remove:
18                 $\forall r \in R_{sd} : \neg h \in A(r)$ 
19          case  $-\partial_{ap}p$ :
20            foreach  $r \in R_{sd} : p \in A(r)$  do
21               $R_{sd} = R_{sd} \setminus \{r\}$ 
22              if  $R_{sd}[C(r)]$  is null then
23                 $\mathcal{S} = \mathcal{S} \cup \{-\partial_{ap}C(r)\}$ 

```

d cannot be proved defeasibly (under ambiguity propagation).

The algorithm first starts by initializing \mathcal{S} with the set of conclusions that are known to be defeasibly true: all the facts and the heads of rules with empty antecedent. Then it iterates on \mathcal{S} until $\mathcal{S} = \emptyset$. Whenever a literal p cannot be proven, those rules with p in their antecedent become inapplicable and thus removed from the theory.

For positive defeasible provability, before inserting $+\partial_{ap}$ into the conclusion set, we have to determine whether the complementary literal is in the supporting set. If the complementary literal is in the supporting set, then the literal is refuted and thus $-\partial_{ap}$ is derived. Otherwise, we have to ensure that all rules with $\neg h$ in their head are either refuted or inapplicable, before deriving the $+\partial_{ap}$ conclusion. Consider the case as shown in example 1. Before inserting d in $+\partial_{ap}$, we have to evaluate whether the complement of d , i.e., $\neg d$, is supported or not. Since the complementary literal ($\neg d$ in this case) is supported, we have to conclude that

³ Defeasible theories with superiority relations and/or defeaters, can be transformed into equivalent theories without superiority relations and defeaters using the techniques described in [9].

The discussion above shows how definite and defeasible conclusions can be derived from a defeasible theory, which is a bit tedious. However, computing the supported/unsupported set of a defeasible theory is very straight forward.

Algorithm 2: Inference algorithm for Support- and Unsupport-set computation

Algorithm: *ComputeSupport*(\mathcal{D})

Data: $\mathcal{D} = (F, R, \emptyset)$: a defeasible theory

Result: Σ^+ - set of supported literals

Result: Σ^- - set of unsupported literals

- 1 $\mathcal{L}^+ = F \cup \{a \in L \mid \exists r \in R_{sd}[a] : A(r) = \emptyset\}$
- 2 $\mathcal{L}^- = \{a \in L \mid R_{sd}[a] \text{ is empty}\}$
- 3 $\Sigma^+ = \emptyset$
- 4 $\Sigma^- = \emptyset$
- 5 **while** \mathcal{L}^+ is not empty **do**
- 6 $\mathcal{L}^+ = \mathcal{L}^+ \setminus \{l\}$ for some $l \in \mathcal{L}^+$
- 7 $\Sigma^+ = \Sigma^+ \cup \{l\}$
- 8 **foreach** $r \in R_{sd} : l \in A(r)$ **do**
- 9 **remove** l from $A(r)$
- 10 **if** $A(r)$ is empty **then**
- 11 $\mathcal{L}^+ = \mathcal{L}^+ \cup \{C(r)\}$
- 12 $R_{sd} = R_{sd} \setminus \{r\}$
- 13 **while** \mathcal{L}^- is not empty **do**
- 14 $\mathcal{L}^- = \mathcal{L}^- \setminus \{l\}$ for some $l \in \mathcal{L}^-$
- 15 $\Sigma^- = \Sigma^- \cup \{l\}$
- 16 **foreach** $r \in R_{sd}, l \in A(r)$ **do**
- 17 $\mathcal{L}^- = \mathcal{L}^- \cup \{C(r)\}$
- 18 $R_{sd} = R_{sd} \setminus \{r\}$

ComputeSupport (algorithm 2) shows how the *support/unsupport* set of a defeasible theory is computed. The idea behind this algorithm is very simple. Whenever there exists a line of reasoning that would lead us to conclude p , we will say that p is supported irrespective of whether its complementary literal, i.e., $\neg p$, is supported or not.

The algorithm is similar to the algorithm we discussed before. It starts by initializing two variables: \mathcal{L}^+ , which stores the set of literals that can be proved definitely, and \mathcal{L}^- , which stores the set of literals that are known to be unprovable. The algorithms then iterate on both sets to derive the supported and unsupported set respectively. That is, for each cycle of the iteration, a positively proved literal will be removed from the bodies of all other rules. Whenever the antecedent of a rule becomes empty, its head will then become a new supported literal for

future iterations. On the other hand, whenever a literal p found to be negatively provable, then all rules with p in their antecedent will be removed from the theory and their heads will be inserted into the unsupported set. These two steps go on until both \mathcal{L}^+ and \mathcal{L}^- become empty.

Executions of the above algorithms can be thought of as execution of transition system on states. As the algorithm proceed, the theory D is simplified and new conclusions are accumulated. The translations for the positive conclusions are based on forward chaining while the negative conclusions are derived by a dual process.

4 Well-Founded Semantics

Well-founded semantics [11] is a fixpoint semantics which was originally developed to provide reasonable interpretation of logic program with negation, but has since been applied to extended logic programs and non-monotonic reasoning. It is a skeptical approximation of answer set semantics such that every well-founded consequences of a logic program P is contained in every answer set of P . Whilst some programs are

not consistent under answer set semantics, well-founded semantics assigns a coherent meaning to *all* programs.

Example 2. Consider the following example:

$$\begin{array}{ll}
 r_1: \Rightarrow doResearch(John) & r_2: doResearch(X) \Rightarrow publishPapers(X) \\
 r_3: publishPapers(X), teachAtUni(X) \Rightarrow professor(X) & r_4: professor(X) \Rightarrow doResearch(X) \\
 r_5: professor(X) \Rightarrow teachAtUni(X) & r_6: teachAtUni(X) \Rightarrow highReputation(X) \\
 r_7: \Rightarrow \neg highReputation(X) & r_6 > r_7
 \end{array}$$

Given a person *John* who does research at university, we would like to ask if John is a *professor*. To derive $professor(John)$ we must derive $publishPapers(John)$ and $teachAtUni(John)$. To derive $teachAtUni(John)$ we need to check $professor(John)$. And we enter in an infinite loop. Consequently neither could we show $highReputation(John)$.

The notion of *unfounded sets* is the cornerstone of well-founded semantics. These sets provide the basis to derive negative conclusions in the well-founded semantics. Intuitively these are collections of literals with no external support. The only way to prove an unfounded set literal is to use literals that are themselves unfounded.

Definition 1. Given a theory \mathcal{D} , its Herbrand base H , and a partial interpretation I , a set $U \subseteq H$ is an unfounded set with respect to I iff each atom $\alpha \in U$ satisfies the following condition: For each instantiated rule R of \mathcal{D} whose head is α one of the following holds:

- Some subgoal of the body is false in I .
- Some positive subgoal of the body occurs in U .

In example 2, the set $\{teachAtUni(John), professor(John), highReputation(John), \neg highReputation(John)\}$ is an unfounded set with respect to the defeasible theory. However, either $highReputation(John)$ or $\neg highReputation(John)$ can be derived if we can remove the loop caused by $professor(John)$ (r_3) and $teachAtUni(John)$ (r_5). Thus only $professor(John)$ and $teachAtUni(John)$ constitute an unfounded set under DL.

4.1 Computing Consequences in DL with Well-Founded Semantic

To nullify evidence DL has to be able to disprove rules [12]. This means that the proof system should be able to demonstrate in a finite number of steps that there is no proof of the rule and thus remove them from the theory. As conclusions cannot be derived using circular arguments, *loops detection* plays a crucial role in deriving conclusions under well-founded semantics. *Failure-by-looping* provides a mechanism for falsifying a literal when it is within a loop with no external support. It helps to simplify a theory by removing inapplicable rules and makes theory becomes decisive, i.e., all rules in the theory are either provable or unprovable [13].

Definition 2. [14] Given a theory \mathcal{D} , let \mathcal{L} be the set of literals appear in \mathcal{D} . Then a loop in \mathcal{D} is a set of literals $L \subseteq \mathcal{L}$ s.t. for any two literals $p_1, p_2 \in L$ there exists a path from p_1 to p_2 in the literal dependency graph of \mathcal{D} all of whose vertices belong to L .

In other words, the subgraph of the literal dependency graph of \mathcal{D} is strongly connected [15]. From the definition of the unfounded set any rule whose head belongs to an unfounded set, or there exists an unfounded literal in their body, is inapplicable. Since unfounded sets are finite, we have the following consequence.

Proposition 1. [15] *Given a theory \mathcal{D} , a partial interpretation I , and unfounded set $U_{\mathcal{D}}$ w.r.t. I . If $U_{\mathcal{D}} \neq \emptyset$, we have $L \subseteq U_{\mathcal{D}}$ for some loop L in \mathcal{D} that is unfounded w.r.t. I .*

The above proposition states that any non-empty unfounded set is a super set of some loop that is itself unfounded. Owing to the fact that loops are bounded above by the *strongly-connected-components (SCC)* in the literal dependency graph, algorithm 3 shows the algorithm used to compute the unfounded set of a defeasible theory.

Algorithm 3: Unfounded set computation

Algorithm: *ComputeUnfoundedSet*(l, \mathcal{D})

Data: l : a literal in \mathcal{L}

Data: $\mathcal{D} = (F, R, \emptyset)$: a defeasible theory

Result: \mathcal{U} : set of unfounded literals in \mathcal{D}

```

1   $\mathcal{U} = \{\emptyset\}$ 
2   $l.id = cnt++$ 
3   $\mathcal{S}.push(l)$ 
4   $\mathcal{P}.push(l)$ 
5  foreach  $r \in R_{sd} : l \in A(r)$  do
6       $c = C(r)$ 
7      if  $c.id == -1$  then
8           $\mathcal{L} = \mathcal{L} \setminus c$ 
9          ComputeUnfoundedSet( $c, \mathcal{D}$ )
10     else if  $c.groupId == -1$  then
11         while  $\mathcal{P}$  is not empty and
12              $\mathcal{P}.top().id > c.id$  do
13                  $\mathcal{P}.pop()$ 
13 while  $\mathcal{P} \neq \emptyset$  and
14      $\mathcal{P}.top().pre > \min(l.pre, -l.pre)$  do
15          $\mathcal{P}.pop()$ 
16 if  $\mathcal{P}.top().id == l.id$  then
17      $\mathcal{P}.pop()$ 
18 else
19     return
19 repeat
20      $t = \mathcal{S}.pop()$ 
21      $t.groupId = gcnt$ 
22      $\mathcal{U} = \mathcal{U} \cup \{t\}$ 
23 until  $\mathcal{S}$  is empty or  $t == l$ 
24  $gcnt++$ 

```

try to the recursive function; and *pops* them (with assigned SCC id) after visited the final member of each SCC. So at the end of the recursive procedure it will return to us all literals encountered since entry that belong to the same SCC, i.e., the set of literals that are unfounded. The algorithm extends the unfounded set through each iteration. That is, to calculate the greatest unfounded set, we have to iterate *ComputeUnfoundedSet* through the set of literals that appear in the defeasible theory \mathcal{D} .

In the algorithm, the designated initial situation is that \mathcal{D} is the simplified defeasible theory s.t. $R_{sd} \neq \{\emptyset\}$ but with *no further conclusions can be derived*; and \mathcal{L} is the set of literals that appear in \mathcal{D} . \mathcal{S} is a stack used to keep track on the set of processed literals; while \mathcal{P} , another stack, containing literals on the search path, is used to decide when to pop the set of SCC from \mathcal{S} . The variables cnt and $gcnt$ store the id of a literal and the group id of the SCC that the literal belongs to respectively. Lastly, \mathcal{U} contains the literals to be extended to an unfounded set.

The algorithm works based on two observations: (1) when we reach the end of the recursive function, we know that we will not encounter any more literals in the same strongly connected set since all literals that can be processed have already been passed (line 5-12); (2) the back links in the graph provide a second path from one literal to another and bind together the SCC (line 13-22).

The algorithm first finds the highest literal reachable based on the literal dependency graph. A literal will be *pushed* onto the stack on en-

As discussed before, failure-by-looping provides a mechanism to remove literals in loops with no external support. That is falsifying all literals in the greatest unfounded set helps to remove loops in the literal dependency graph of a defeasible theory, and thus the well-founded model of the theory can be derived subsequently.

5 Related works

A number of defeasible logic reasoners and systems have been proposed in the recent year to cover well-founded variants as well as other intuitions of non-monotonic reasoning. The approaches to define a well-founded variant can be classified under three categories: failure-by-loop [1,16,13]; unfounded set [17]; translation to other formalism or extended logic programming [18,6,19].

Nute [2] was the first to propose a form of loop-to-failure, but he did not related the logic to well-founded-semantics, and it was limited to a basic variant, and no implementation was proposed. [16] extended the work, and related the variant to well-founded defeasible logic, but they claim that the resulting variant is ambiguity propagation, and those two notions are entangled. [12] adopts a more sophisticated failure-by-loop algorithm for a clausal variant of defeasible logic (with the consequent increase of the computational complexity).⁴

The unfounded set approach on which the current paper is based was proposed in [17] where a bottom-up approach was presented. The bottom-up approach led to metaprogram representation of defeasible logic and defeasible theories, with the consequent development of the family of defeasible logic framework [20,10].

The meta-program approach is at the foundation of the approaches based on transformation in other formalisms. DR-Prolog [19] provides a Semantic Web enabled implementation of defeasible logic. DR-Prolog directly implements the meta-programs of [20], covering thus various variants, and the well-founded variants are obtained by invoking a well-founded version of a Prolog interpreter. The system is query based and does not compute the extension of a theory.

Other logic formalisms, such as CLIPS or other extended logic programs have been used to implement some variants of the DL [18,6] so that conclusions can be drawn using the underlying reasoning engine. However, most of the systems are query based and do not compute directly the extension of a theory. However transformation based approach may lead to some counterintuitive results [3] as, in most cases, the representational properties of defeasible logic cannot be captured correctly.

6 Conclusions

This paper presents algorithms for computing the consequences of the ambiguity propagation variant and well-founded semantics of defeasible logic. It contributes to the computational aspect of the two variants in a practical approach such that consequences of both variants (as well as their combination) can be computed with linear complexity,

⁴ See <http://www.cit.griffith.edu.au/~arock/defeasible/Defeasible.cgi> for a query based reasoner implementing various variants.

which make DL a possible candidate for some computational demanding jobs, or tasks that require immediate response, such as reasoning on the Semantic web.

Recently [8] has studied several variants of defeasible logic based on their abstract presentation of the proof theory. However, the relations between the ambiguity propagation variant, the well-founded semantics, and the well-founded variants of ambiguity propagation is still unclear and further investigation is needed.

References

1. Nute, D.: Defeasible logic. In Gabbay, D., Hogger, C., eds.: *Handbook of Logic for Artificial Intelligence and Logic Programming*. Volume III. Oxford University Press (1994) 353–395
2. Nute, D.: Defeasible logic. In: *14th International Conference on Application of Prolog, IF Computer Japan* (2001) 87–114
3. Antoniou, G.: A Discussion of Some Intuitions of Defeasible Reasoning. In: *Methods and Applications of Artificial Intelligence*. Springer (2004) 311–320
4. Maher, M.J.: Propositional defeasible logic has linear complexity. *Theory and Practice of Logic Programming* **1**(6) (2001) 691–711
5. Maher, M.J., Rock, A., Antoniou, G., Billington, D., Miller, T.: Efficient defeasible reasoning systems. *International Journal on Artificial Intelligence Tools* **10**(4) (2001) 483–501
6. Bassiliades, N., Antoniou, G., Vlahavas, I.: A defeasible logic reasoner for the semantic web. *International Journal of Semantic Web and Information Systems (IJSWIS)* **2**(1) (2006) 1–41
7. Lam, H.P., Governatori, G.: The making of SPINdle. In Paschke, A., Governatori, G., Hall, J., eds.: *RuleML 2009*, Springer (2009) 315–322
8. Billington, D., Antoniou, G., Governatori, G., Maher, M.J.: An inclusion theorem for defeasible logic. *ACM Transactions in Computational Logic* **to appear** (2010)
9. Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: Representation results for defeasible logic. *ACM Transactions on Computational Logic* **2**(2) (2001) 255–286
10. Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: A flexible framework for defeasible logics. In: *AAAI-2000*, AAAI/MIT Press (2000) 401–405
11. Van Gelder, A., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. *J. ACM* **38**(3) (1991) 619–649
12. Billington, D.: Propositional clausal defeasible logic. In Hölldobler, S., Lutz, C., Wansing, H., eds.: *JELIA*. Volume 5293 of *LNCS.*, Springer (2008) 34–47
13. Billington, D.: A plausible logic which detects loops. In: *NMR 2004*. (2004)
14. Lee, J.: A model-theoretic counterpart of loop formulas. In: *IJCAI'05*. (2005) 503–508
15. Anger, C., Gebser, M., Schaub, T.: Approaching the core of unfounded sets. In: *NMR 2006*. (2006) 58–66
16. Maier, F., Nute, D.: Well-founded semantics for defeasible logic. *Synthese* (2008)
17. Maher, M.J., Governatori, G.: A semantic decomposition of defeasible logics. In: *AAAI'99*. (1999) 299–305
18. Madalińska-Bugaj, E., Lukaszewicz, W.: Formalizing defeasible logic in cake. *Fundam. Inf.* **57**(2-4) (2003) 193–213
19. Antoniou, G., Bikakis, A.: DR-Prolog: A system for defeasible reasoning with rules and ontologies on the semantic web. *IEEE Trans. Knowl. Data Eng.* **19**(2) (2007) 233–245
20. Antoniou, G., Billington, D., Governatori, G., Maher, M.J., Rock, A.: A family of defeasible reasoning logics and its implementation. In: *ECAI 2000*. (2000) 459–463