Extending Defeasible Prolog: ‘Even-if’, Preemption, and Defeasible Reasoning

Ahti-Veikko Pietarinen[⋆]

Department of Philosophy, P.O. Box 9, FIN-00014 University of Helsinki
`pietarin@cc.helsinki.fi`

Abstract. Defeasible Prolog (d-Prolog) is a Prolog metainterpreter designed by Donald Nute to implement nonmonotonic inference based on a system of defeasible logic. Defeasible logic promotes enthymemic reasoning on incomplete set of premisses, retracted on the presence of contrary information. In this paper, it is shown how to give proof conditions for the ‘even-if’ conditions of defeasible logic. This is done by allowing the pre-emption of defeaters, in other words preventing some rules of rebutting other, specific rules. These proof conditions are then implemented to d-Prolog. Some computational results are presented for the given examples. The nature and goals of defeasible reasoning is also assessed.

1 Introduction

In Donald Nute’s system of defeasible reasoning [7–9], the goal is to give a formalised account of nonmonotonic defeasible inferences like “typically, φ’s are ψ”, “normally, φ’s are ψ”, “usually, φ’s are ψ”, or perhaps “reasonable grounds for holding φ warrant reasonable ground for holding ψ”. These inferences hold only if a defeasible theory does not contain additional rules that represent contrary information. Such contrary information may lead to the conclusion that the state of affairs exemplified by the previous set of defeasible rules no longer holds. In the presence of such rules, previous inferences may become defeated, and so defeasible reasoning exhibits nonmonotonicity: a set of conclusions does not always grow monotonically when new premises are added.

Defeasible Prolog (d-Prolog) is a Prolog metainterpreter, designed by Donald Nute [10] to implement some basic forms of nonmonotonic defeasible inference. It has two rules to represent defeasibility, defeasible implications and defeaters. In order to represent defeasible implication (ψ `:=` Φ, read “typically, Φ’s are ψ”) as well as the defeater (ψ `:^` Φ, read “if Φ, it can be taken as a reason to doubt that not ψ” or “if Φ, it might be that ψ”), new two-place operators “`:=`” and “`:^`” are added to the usual syntax of Prolog. Ordinary Prolog implication “`:-`” has, however, its usual meaning.

Although it is not clear what the intuitive difference between two rules is, their uses in the proof derivations are formalised slightly differently. A defeater

⋆ Supported by the Academy of Finland (Grant No: 103130).

does not give evidence to support some conclusion, since its intended purpose is just to interfere with the derivations from the defeasible rules.

In d-Prolog, rule heads are literals (atomic sentences or their negations), and bodies are conjunctions of atomic sentences. Unlike ordinary Prolog with negation-as-failure, literals, and hence also the heads of the rules may be explicitly negated with a one-place predicate "`neg`". The approach Nute advocates in d-Prolog is entirely proof theoretic.

A defeasible theory $\mathcal{T}$ is a tuple $\langle K, R\rangle$, where K is a finite set of literals and R is a finite set of rules. A proof of a literal ψ is defined as a proof tree t, where ψ is a root node of t and t is a finite labelled tree such that for every node n of t there is a theory $\mathcal{T}$ and some literal which is flagged positive (φ^+) or negative (φ^-). When the node contains φ^+, we know that the literal is defeasibly derivable from $\mathcal{T}$ with respect to the set of proof conditions Σ ($\mathcal{T} \vdash_\Sigma \psi$); on the other hand, if the node has the label ψ^-, it indicates that the literal is demonstrably not derivable from the theory $\mathcal{T}$ by the conditions in Σ ($\mathcal{T} \nvdash_\Sigma \psi$).

Defeasible logic is defined as a set of conditions Σ on every node of the tree t for which it is never the case that $\mathcal{T} \vdash_\Sigma \psi$ and $\mathcal{T} \nvdash_\Sigma \psi$. To solve the problem of which defeasible rules should be used to defeat other defeasible rules, an explicit partial order (superiority relation) is defined between defeasible rules. If a rule r_1 is superior to a rule r_2 ($r_1 \sqsupset r_2$), it may be used to defeat the inferior rule r_2.

In some cases, it is also possible to extract information about the partial order by using the method of more specific antecedent [7]. By defining superiority by antecedent specificity, the antecedent conditions of one rule are derivable from the antecedent conditions of another rule. However, an extra complication ensues, for some parts of the derivations in a proof tree may be based on just the subtheory of $\mathcal{T}$. Therefore, some labellings may vary on their admissible set of literals and rules upon which the proofs of literals in rules defined by more specific antecedents depend.

Different rules interact with each other in the following ways. The rule ψ `:=` Φ is defeated by the rule η `:=` X or η `:-` X, if the head η is either a negation of ψ or some literal which is explicitly stated as incompatible with ψ. Such a literal η is contrary to ψ. Next, the rule ψ `:=` Φ is undercut by the defeater η `:^` X, if η is contrary to ψ and ψ `:=` Φ is not superior to η `:^` X. A defeasible rule ψ `:=` Φ or a defeater ψ `:^` Φ may also be pre-empted, whenever there exists a literal η contrary to ψ which is derivable from the theory, or there is an ordinary Prolog rule η `:-` X with a contrary head, or for the defeasible rule ψ `:=` Φ there is a rule η `:=` X, whose head η is contrary to ψ and which is superior to it, that is, if (η `:=` X) $\sqsupset$ (ψ `:=` Φ).

Moreover, the literals contained in the bodies X in any of the former rules must always be defeasibly derivable from the defeasible theory $\mathcal{T}$. Preemption relaxes the assumption that applicable defeasible rules to defeat competing contrary rules must always be superior to every other competing rule. When preemption is enabled, once the defeasible rule is defeated it loses its capacity to defeat any other rule. Unfortunately, pre-emption is computationally costly.

Starting with the d-Prolog of [10], in the present paper we extend d-Prolog with the so called even-if rules and even-if proof conditions specified to those rules. Even-if rules and even-if conditions without pre-emption were given in [8], but they have not been implemented in d-Prolog. Our task is, therefore, first to formulate these conditions at the theoretical level such that they also enable pre-emption, and then show how they may be implemented in d-Prolog. Finally, some general points concerning defeasible reasoning and its implementation are discussed, including the possibility of stratifying d-Prolog programs.

2 Rules for the Even-If Conditions

In addition to the basic method of defeasible reasoning, Nute has introduced even-if rules and conditions in his system of defeasible logic [8]. The aim of these rules is to extend basic defeasible reasoning that allows new kinds of inference. Formally, an even-if rule is a rule ψ `:=` Ξ `|` Φ, and is read "if Φ, then (typically) ψ even if Ξ holds". Let us call the condition Φ the main condition and the condition Ξ the even-if condition. These rules are always defeasible, that is, they do not apply to the ordinary Prolog rules. Moreover, they permit the superiority relation to be defined by the method of finding a more specific antecedent. Conditions for the proper use of these rules have not, however, been given in a form that enables the pre-emption of defeaters.

To illustrate the use and applicability of even-if rules, consider following two examples.

Example 1. A beneficiary is suspect:

```
suspect(X) := beneficiary(X).
```

With an alibi a person usually is not, however, suspect even if he is a beneficiary:

```
neg suspect(X) := alibi(X) | beneficiary(X).
```

Tom has an alibi: `alibi(tom)`. Clearly he must not be a suspect. In both cases it can, indeed, be concluded tentatively that `neg suspect(tom)` is a derivation from the theory $\mathcal{T}$.

Example 2. Let us add to the previous theory the defeater, according to which a person with an alibi who has not provided reliable documents (or has forged them) may very well be suspect:

```
(suspect(X) :^ (alibi(X), neg reliable_documents(X)).
```

If Tom has not provided reliable documents

```
neg reliable_documents(tom),
```

it is difficult to conclude, with this amount of information, whether Tom is suspect or not. Perhaps the best one may hope is to draw no conclusion. In the extension of d-Prolog, a fragment of which is given in Appendix, both `neg suspect(tom)` and `suspect(tom)` are demonstrably not derivable from the $\mathcal{T}$, since the rule

```
suspect(X) :^ (alibi(X), neg reliable_documents(X))
```

is not an acceptable rule to pre-empt an inferior rule

```
neg suspect(X) := alibi(X) | beneficiary(X).
```

This is because pre-emption is only possible if the rule is defeated with a superior rule.

Conditions that are responsible for the correct even-if derivations emerge from suitable modifications to the conditions in [8] in the following respects. First, a substitution is removed for simplicity.[1] Second, the notation will be changed and made uniform, since we deal with d-Prolog syntax. Third, the pre-emption of defeaters is taken into account by adding some extra constraints.

Let n be an arbitrary node of the proof tree t, K the set of literals, R the set of rules, and let the $@\psi$ denote the case when ψ is a tentative conclusion derived from the theory $\mathcal{T} = \langle K, T\rangle$. The following conditions P^+ and P^- are defined for all the nodes n, the former for the literals that are defeasibly derivable and the latter for the literals that are demonstrably not derivable. We also require the main conditions to be nonempty.

P^+: The node n is labelled $\langle K, R, @\psi^+\rangle$, if either n has a child labelled $\langle K, R, \texttt{neg } \psi^-\rangle$ or there exists a rule ψ `:=` Ξ `|` $\Phi \in R$ such that all of the following (1), (2) and (3) hold:

- (1) for every $\varphi \in \Phi$ a node n has a child labelled $\langle K, R, @\varphi^+\rangle$ (= every literal in the main body is derivable)
- (2) for every `neg` ψ `:=` $H \in R$ there exists $\eta \in H$ and a child of n labelled $\langle K, R, @\eta^-\rangle$ (= every contrary rule must have a nonderivable literal in the body)
- (3) for every `neg` ψ `:=` Δ `|` $\Theta \in R$ or `neg` ψ `:^` $\Theta \in R$ either (a) or (b) holds:
  - (a) there exists $\theta \in \Theta$ and a child of n labelled $\langle K, R, @\theta^-\rangle$ (= every contrary rule must have a nonderivable literal in the main body)
  - (b) all of the following (i), (ii) and (iii) hold:
    - (i) there exists $\varphi \in \Phi \cup \Xi$ and a child of n labelled $\langle \Theta, R, @\varphi^-\rangle$ (= specificity: some literal in the antecedent conditions of the main rule is not derivable from the antecedent conditions of the contrary rules)
    - (ii) for every $\theta \in \Theta$ there exists a child of n labelled $\langle (\Phi \cup \Xi), R, @\theta\varphi^+\rangle$ (= specificity: every literal in the main condition of the contrary rule is derivable from the antecedent conditions of the main rule)
    - (iii) for every $\varphi \in \Phi \cup \Xi$ there exists a child of n labelled $\langle K, R, @\varphi^+\rangle$ (= preemption: every literal in the body of the main rule is defeasibly derivable).

P^-: The node n is labelled $\langle K, R, @\psi^-\rangle$, if either n has a child labelled $\langle K, R, @\psi^+\rangle$ or both (1) and (2) hold:

[1] Substitution refers to quantified defeasible logic that has been sketched in [8].

(1) for every ψ `:=` $\Phi \in R$, either (a) or (b) holds:
  (a) there exists $\varphi \in \Phi$ and a child of n labelled $\langle K, R, @\varphi^- \rangle$
  (= a literal in the body is demonstrably not derivable)
  (b) there exists ψ `:=` $H \in R$ such that for every $\eta \in H$ a node n has a child labelled $\langle K, R, @\eta^+ \rangle$
  (= some other rule with the same head has a derivable body)

(2) for every ψ `:=` Ξ `|` $\Phi \in R$, either (a), (b) or (c) holds:
  (a) there exists $\varphi \in \Phi$ and a child of n labelled $\langle K, R, @\varphi^- \rangle$
  (= a literal in the head is demonstrably not derivable)
  (b) there exists `neg` ψ `:=` $X \in R$ such that for every $\chi \in X$ a node n has a child labelled $\langle K, R, @\chi^+ \rangle$
  (= some contrary rule has a derivable body)
  (c) there exists `neg` ψ `:=` Δ `|` $\Theta \in R$ or `neg` ψ `:^` $\Theta \in R$ such that for every $\theta \in \Theta$ a node n has a child labelled $\langle K, R, @\theta^+ \rangle$, and either (i), (ii) or (iii) holds:
    (i) there exists $\theta \in \Theta$ and a child of n labelled $\langle (\Phi \cup \Xi), R, @\theta^+ \rangle$
    (= specificity: some literal in the main body of the rule is derivable from the body of the original rule)
    (ii) for every $\varphi \in \Phi \cup \Xi$ a node n has a child labelled $\langle \Xi, R, @\varphi^+ \rangle$
    (= specificity: every literal in the body of the original rule is derivable from the main body of the contrary rule)
    (iii) there exists $\varphi \in \Psi \cup \Xi$ and a child of n labelled $\langle K, R, @\theta^- \rangle$.
    (= pre-emption: some literal in the body of the original rule is demonstrably not derivable).

Let **M** be a monotonic core of a defeasible logic consisting of the four basic conditions, and $\mathbf{SS}^+$ is a semi-strictness condition [7]. In terms of the semistrictness condition, an ordinary rule ψ `:-` Φ may defeat another ordinary rule `neg` ψ `:-` Ψ if the antecedent Ψ is only defeasibly derivable.

On the depth of the proof tree t it can now be shown that there is no theory $\mathcal{T}$ and a literal ψ such that $\mathcal{T} \vdash_{\mathbf{P}} \psi$ and $\mathcal{T} \not\vdash_{\mathbf{P}} \psi$. This establishes that $\mathbf{P} = \mathbf{M} \cup \{SS^+, P^+, P^-\}$ is a defeasible logic.

3 Extending d-Prolog

In order to incorporate previous conditions into d-Prolog, we add to its syntax a new two-place relation "`|`". Its aim is to distinguish between the primary condition Φ in the rule ψ `:=` Ξ `|` Φ and the secondary even-if condition Ξ, as in conditions P^+ and P^-.[2] We need to consider the following three modifications and additions to the defeating and undercutting conditions of [10].

1. The rule ψ `:=` Ξ or ψ `:-` Ξ is defeated (as implemented in the predicate `defeated/2`), if there exists an even-if rule φ `:=` Ξ `|` Φ the head φ of which is contrary to ψ, the body Φ of which (without the even-if condition Ξ) is defeasible derivable, and the even-if condition Ξ of which is a body of such a rule r that is not superior to the rule φ `:=` Ξ `|` Φ.

[2] Note that in the appendix, the symbol '`|`' is replaced with the symbol '`#`'.

2. The rule ψ `:=` Ξ `|` Φ is defeated
   - if there exists a rule φ `:=` Θ the head φ of which is contrary to ψ, Θ is defeasibly derivable, and the condition Ξ is not the head of any defeasible rule that is not superior to the rule φ `:=` Θ, or
   - if there exists a rule φ `:=` Φ `|` Θ, the head φ of which is contrary to ψ, Θ is defeasibly derivable, and the even-if condition Φ is the head of a defeasible rule r that is not superior to the rule φ `:=` Φ `|` Θ.
3. The rule ψ `:=` Ξ `|` Θ is undercut (`undercut/2`), if there exists a defeater φ `:^` Θ, the head of which is contrary to ψ, Θ is defeasibly derivable, and ψ `:=` Ξ `|` Θ is not superior to it.

The preemption of defeaters (`preempted/2`) is similar to the defeating conditions (Appendix A).

There are also two new definitions for the defeasible derivability (`def_der/2`), three definitions for defining superiority relation with the defeasible specificity (`sup_rule/2`) and some other definitions for syntax and occurrences of the even-if rules. Some of these predicates are described in the appendix. We omit further details here.

4 Computational Considerations

To get a glimpse of the performance of defeasible inferences in d-Prolog, several runs have been performed as regards typical benchmark problems of nonmonotonic reasoning. The results for the examples 1 and 2 in the extended d-Prolog are summarised in Table 1. Note how enabling the preemption of defeaters increases the sizes of proof trees. In general, the sizes are enormous, and so defeasible reasoning of this kind is far from tractable.

pre-	Example 1		Example 2	
empt	`@ suspect(tom)`	`@ neg suspect(tom)`	`@ suspect(tom)`	`@ neg suspect(tom)`
+	23	90	23	138
–	53	98	53	270

Table 1. Some sizes of proof trees for the queries '`@ suspect(tom)`' and '`@ neg suspect(tom)`' for the predicates presented in examples 1 and 2.

As the first approximation, there are some straightforward strategies to limit the complexity of defeasible reasoning. For example, the sizes of the proof trees can be restricted to simplify inferences in the following two ways:

- Restrict the depth of the proof tree by generating paths only up to the certain fixed limit. After reaching the limit, the proof backtracks to the earlier nodes.
- Restrict the branching factor of the proof tree by keeping the size of the rule set R small and also the maximum number of conjunctions in every rule reasonably small.

The former method, however, has a notorious side effect: since the nature of the 'argumentative' defeasible inference is depth-first search (some labels may depend on the other distant labels), constraining it may cause the horizon effect. Moreover, limiting maximal depth amounts to the cheap method of loop checking.

The latter method is hence more appropriate, since it reflects the nature of defeasible reasoning better. In argumentation, only the most pertinent and plausible arguments should be used to support arguments. In conversational situations, it is often necessary, or at least pragmatic, to lay size restrictions to the allowable set of one's assertions. Usually only one or two defeaters, or objections, ought to be enough to render opposite views unwarranted.

Further emendations are possible. In defeasible logic, derivation of incompatible clauses and prevention of cycles lies in the hands of the programmer. Defeasible logic cannot detect them. However, whenever there is a fear of cyclic dependency graphs, a possibility is to stratificate the predicates so that they do not depend on each other in a circular manner. By stratified sets of rules it is possible to solve precisely which rules one should use to derive a conclusion. The idea of stratification is that, prior to referring to a negation of a fact, the fact itself needs to be defined. (This idea has immediate cognitive plausibility: to be able to think of a negated concept the concept itself has to be accessible to the thought, albeit with certain neural inhibition.) In terms of logic programming, a stratified program has a partition of its clauses into hierarchical sets in which negative goals of clauses are defined in lower-level predicates.

What if some programs cannot be stratified? The proposed solution in the well-founded semantics is that some facts receive the third truth-value of 'undefined'. In the stable model semantics the logic remains bivalent, and non-stratified programs may have several models or none. Typically, however, stratification amounts to programs that have a model and do not contain mutually dependent predicates.

Furthermore, it is possible to consider different grades of stratification in the sense that only certain kinds of rules, such as defeasibe rules, strict rules or defeaters, are subject to stratification, amounting to semistratified programs.

To have stratification is an obvious future step that needs to be done to improve the performance of d-Prolog. Before that, however, a feasible semantics has to be defined for defeasible logic and a fortiori for d-Prolog. This is not our concern in this paper, but [6, 12] have provided steps to that direction.

5 Conclusions and Related Work

In this paper, it was shown how to give proof conditions for defeasible 'even-if' rules, by allowing preemption, namely preventing some rules of rebutting other, specific rules. These proof conditions were implemented in d-Prolog.

Some comments that put defeasible logic into a wider perspective are in order.

The aim of defeasible logic is to model aspects of human reasoning in which classical, monotonic logic falters. However, without proper semantics it is in the end unclear how far such goals will reach. But rules of the above systems are constitutive. That the system produces correct results for some sample of cases do not tell what these rules mean. For instance, proof theory does not tell what the intended meaning of defeasible implication is, in other words what it is for it to be satisfied in a given interpretation.

In the present paper, the sole reason for formulating further rules of defeasible inference in the spirit of proof theory is that they are easier to implement. However, the defeasible systems are no longer effective. What is the purpose of these rules?

In nonmonotonic systems, there is a trade-off between maximising the set of correct conclusions and minimising the set of incorrect ones. It is not plausible to think that the latter set will be empty. In this sense, the rules are no longer entirely constitutive, and there is a strategic element in them.

Furthermore, one may think that for any set of compatible examples of defeasible inference, there exists a defeasible logic that produces correct results for any single set of examples. This may be done by tweaking the constitutive component of rules indefinitely. Even so, the problem of the meaning of the rules remains unanswered. For instance, it is not clear, due to the lack of model theory, what the motivations are to separate defeasible rules and the defeaters from each other. They are ambiguous, subordinate to the kind of meaning holism that subdues their differences.

Models of defeasible logic should depict situation, states, or classes of objects that are in some plausible sense typical, *ceteris paribus*, and insensitive to vagueness of natural language predicates, in the sense of not giving rise to any 'fuzzy' system of graded truth values. Prototype theory [5, 15] aims at clarifying the contribution that new information brings to pre-existing structures of knowledge (schemas, frames, scripts etc.) by relaxing the strict category membership or the class-inclusion relation. Prototype theory thus appears to provide a promising candidate for the further development of a defeasible model theory.

Until a proper model theory is at hand, defeasible logic will remain enthymemic. In other words, it is an argumentative rather than a logical system, in which reasoning proceeds in terms of incomplete sets of premisses. Essentially, then, it functions by way of common knowledge that the agent, engineerer or modeller brings in. It thus appeals to the rhetoric ideas associated with programming (such as *ethos* and *pathos*) rather than strictly logical semantics.

A recent proposal for the workable semantics of defeasible logic is found in [6]. In that semantics, models are taken to be 'states of minds' that differentiate between necessary (definite) reasoning in terms of knowing a proposition and defeasible reasoning in terms of believing a proposition. It reflects the well-founded semantics in that there may be propositions that are neither known, believed, not known nor not believed. What this agnostic interpretation produces is an innovative epistemic approach to defeasible logic, replacing an objective concept of a model with a cognitive one. This is fine as far as it goes, but regrettably, it does not answer, among other things, the central question of the intended difference in meaning between defeasible implication and the defeater. The credulous fixed-point semantics (exemplifying, so to speak, rhetoric *pathos*) considered in [12] does not tell the difference, either.

Another line of development intergrated defeasible rules of logic programming with argumentative structures and defeasibility among arguments [4] in legal, economic and decision-making systems [14], both of which derive their inspiration from much of the same source of defeasibility in semantic networks [13].

Whilst d-Prolog underwent a protracted improvement over a period of many years, other paradigms entered the scene. Answer-set programming is a dominant paradigm to perform nonmonotonic reasoning [1, 2], including defeasible rules. More work needs to be done in that direction.

As noted, proper logical semantics still awaits to be defined for a sufficiently comprehensive (stratified) d-Prolog. Likewise, the deontic versions of defeasibility [11] are likely to benefit from similar revisions.

References

1. Brewka, G., and Eiter, T., 1999. Preferred answer sets for extended logic programs. *Artificial Intelligence* 109, 297–356.
2. Brewka, G., and Eiter, T., 2000. Prioritizing default logic. In Hölldobler, S. (ed.), *Intellectics and Computational Logic*. Kluwer, Dordrecht, 27–45.
3. Gabbay, D., Hogger, C.J., and Robinson, J.A., eds, *Handbook of Logic in Artificial Intelligence and Logic Programming: Nonmonotonic Reasoning and Uncertain Reasoning*. Clarendon Press, Oxford, 1994.
4. García, A.J., and Simari, G.R., 2004. Defeasible logic programming: an argumentative approach. *Theory and Practice of Logic Programming* 4, 95–138.
5. Lakoff, G., 1987. *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind*. Chicago University Press, Chicago.
6. Maher, M., 2002. A model-theoretic semantics ofr defeasible logic. In Decker, H., et al., eds., *Proc. PCL 2002*.
7. Nute, D., 1992. Basic defeasible logic. In Fariñas del Cerro, L., and Penttonen, M., eds, *Intensional Logics for Logic Programming*. Oxford University Press, Oxford, 125–154.
8. Nute, D., 1994a. A decidable quantified defeasible logic. In Prawitz, D., Skyrms, B. and Westerståhl, D., eds., *Logic, Methodology and Philosophy of Science* 9, Elsevier Science, Holland, 263–284.
9. Nute, D., 1994b. Defeasible logic. In [3, 353–394].
10. Nute, D., 1997a. Defeasible Prolog. In Convington, M., Nute, D., and Vellino, A., *Prolog Programming in Depth*, 2nd ed., Englewood Cliffs, NJ: Prentice Hall.
11. Nute, D., ed., 1997b. *Deontic Defeasible Logic*. Dordrecht: Kluwer.
12. Nute, D., 2001. Defeasible logic, manuscript.
13. Pollock, J., 1987. Defeasible reasoning. *COgnitive Science* 11, 481–518.
14. Prakken, H., and Wreeswijk, G., 2002. Logical systems for defeasible argumentation. In Gabbay, D., and Guenthner, F., ed., *Handbook of Philosophical Logic* , 2nd ed., Kluwer, Dordrecht, 219–318.
15. Rosch, E., 1973. Natural categories. *Cognitive Psychology* 4, 328–350.

A Clauses to Extend d-Prolog

```
init :- op(1100,fx,@), % defeasible conclusion
        op(900,fx,neg), % negation
        op(1100,xfy,:=), % defeasible implication
        op(1100,xfy,:^), % defeater
        op(1100,xfy,#). % even-if

:- dynamic((neg)/1, (:=)/2, (:^)/2, (#)/2).
:- multifile((neg)/1, (:=)/2, (:^)/2, (#)/2).

def_der(KB,Goal) :-
        preemption,
        def_rule(KB,(Goal := (ConditionX # Condition))),
        \+ (contrary(Goal,Contrary1),
             strict_der(KB,Contrary1)), def_der(KB,Condition),
        \+ (contrary(Goal,Contrary2),
             clause(Contrary2,Condition2),
             Condition2 \== true,
             def_der(KB,Condition2)),
        \+ (contrary(Goal,Contrary3),
             def_rule(KB,(Contrary3 := (Condition4 # Condition3))),
             def_der(KB,Condition3),
             \+ (preempted(KB,(Contrary3 := (Condition4 # Condition3))))),
        \+ (contrary(Goal,Contrary5),
             def_rule(KB,(Contrary5 := Condition5)),
             def_der(KB,Condition5),
             \+ (preempted(KB,(Contrary5 := Condition5)))),
        \+ (contrary(Goal,Contrary6), (Contrary6 :^ Condition6),
             def_der(KB,Condition6),
             \+ (preempted(KB,(Contrary6 :^ Condition6)))).

defeated(KB,(Head := (Body1 # Body))) :-
        contrary(Head,Contrary),
        def_rule(KB,(Contrary := (Condition2 # Condition))),
        def_der(KB,Condition),
        \+ sup_rule((Head := (Body1 # Body)),
                         (Contrary := (Condition2 # Condition))),!.

undercut(KB,(Head := (Body1 # Body))) :-
        contrary(Head,Contrary),
        (Contrary :^ Condition), def_der(KB,Condition),
        \+ sup_rule((Head := (Body1 # Body)),(Contrary :^ Body)),!.

preempted(KB,(Head := (Body1 # Body))) :-
        contrary(Head,Contrary),
        def_rule(KB,(Contrary := (Condition2 # Condition))),
        def_der(KB,Condition),
        sup_rule((Contrary := (Condition2 # Condition)),
                          (Head := (Body1 # Body))),!.
```