Defeasible Logic

Donald Nute

Department of Philosophy and Artificial Intelligence Center,
The University of Georgia, Athens, GA 30605, U.S.A.,
dnute@uga.edu

Abstract. We often reach conclusions partially on the basis that we do not have evidence that the conclusion is false. A newspaper story warning that the local water supply has been contaminated would prevent a person from drinking water from the tap in her home. This suggests that the absence of such evidence contributes to her usual belief that her water is safe. On the other hand, if a reasonable person received a letter telling her that she had won a million dollars, she would consciously consider whether there was any evidence that the letter was a hoax or somehow misleading before making plans to spend the money. All to often we arrive at conclusions which we later retract when contrary evidence becomes available. The contrary evidence *defeats* our earlier reasoning. Much of our reasoning is *defeasible* in this way. Since around 1980, considerable research in AI has focused on how to model reasoning of this sort. In this paper, I describe one theoretical approach to this problem, discuss implementation of this approach as an extension of Prolog, and describe some application of this work to normative reasoning, learning, planning, and other types of automated reasoning.

1 Introduction

An important goal in reasoning is to reach only true conclusions. But we also require that our conclusions be justified. These two constraints are not the same; a justified belief can be false, and a true belief can be unjustified. Although true beliefs are our goal, we have no guarantee of truth beyond the justification we have for our beliefs. If the methods we use to reach new conclusions based on old conclusions preserve truth, then once we reach a conclusion we cannot later reject it because of new information. We could only reject an earlier conclusion if we rejected some reason it was based on. We call reasoning of this sort *monotonic*. Human reasoning is not and should not be monotonic. We often reject old conclusions based on new evidence, even when those old conclusions were justified by the evidence we had at the time we arrived at them. Justification preserving reasoning is *not* monotonic.

Why should human reasoning be nonmonotonic? Because monotonic reasoning is too restrictive. Monotonic reasoning in general and truth preserving reasoning in particular work primarily to prevent us from reaching false conclusions. It only allows us to reach conclusions that we could not possibly doubt so long as our original reasons remain intact. It can be dangerous to believe things

O. Bartenstein et al. (Eds.): INAP 2001, LNAI 2543, pp. 151–169, 2003.

that are false, but it can be just as dangerous not to believe things that are true. We need a reasoning system that lets us draw likely conclusions with less than conclusive evidence. And we need a mechanism for correcting this kind of reasoning in light of further evidence.

The *absence* of information can sometimes be a *positive* reason for believing something. Is there any milk in the refrigerator? We look and we do not see any milk. The failure to find evidence of milk in this case is a good reason to believe that there is no milk in the refrigerator. For another example, I believe that there is a cat in front of me. I believe this because there *appears* to be a cat in front of me. That seems to be ample evidence. Of course, we can think of situations where I would be wrong. I might be hallucinating, or there might be a holograph of a cat, or there might be a mirror and the cat I think I see in front of me is actually behind me. But I have no reason to believe that I am hallucinating, and there is no evidence of a holographic projector or of a mirror. The absence of evidence that my perceptual circumstances are abnormal provides part of the justification for my belief that there is a cat in front of me. If I knew I was in a situation where holographic images were likely, I might not even form the belief that there is a cat in front of me. This is not reasoning with uncertainty in the normal sense. It is recognizing that the absence of information can help justify our beliefs.

I am going to describe a particular approach to nonmonotonic reasoning. I will discuss the formal theory behind this approach, work that has been done on implementing this theory, and some applications that have been developed using this approach to nonmonotonic reasoning.

2 Defeasible Logic and Its Language

Suppose that $\vdash$ is the consequence relation of some formal system Σ. Then Σ is monotonic just in case for any two sets S and T of formulas and any formula ϕ of the language of Σ, if $T \vdash \phi$, then $T \cup S \vdash \phi$. We noted earlier that any reasoning system that preserves truth must be monotonic, butt a reasoning system that preserves *justification* will *not* be monotonic. That means that a belief that ϕ might be justified based on our belief in some other set of propositions S, but there could be a set of propositions T such that if we came to believe all the propositions in $S \cup T$ we would no longer be justified in believing ϕ.

Many nonmonotonic formalisms use a special unary operator to mark peculiar conditions in rules. As long as all the conditions for the rule are satisfied, the consequent of the rule is detachable. For example, in autoepistemic logic ([1]) we can always derive χ from ϕ, $\sim K \sim \psi$, and $\phi \wedge \sim K \sim \psi \supset \chi$. By contrast, defeasible systems use rules whose consequents may not be detachable even when their antecedents are derivable ([2,3,4,5,6,7,8]). Detachment of the consequent of one of these *defeasible* rules may be *defeated* by a fact or another rule. The conflicting rule may either *rebut* the first rule by supporting a conflicting consequent, or it may simply *undercut* the first rule by identifying a situation in which the rule does not apply ([6]). Defeasible logic uses strict rules, defeasible rules, and undercutting defeaters.

We define atomic formulas in the usual way. A literal is any atomic formula or its negation. All and only literals are formulas of our language. Where ϕ is an atomic formula, we say ϕ and $\sim\phi$ are the complements of each other. $\neg\phi$ denotes the complement of any formula ϕ, positive or negative.

Rules are a class of expressions distinct from formulas. Rules are constructed using three primitive symbols: $\rightarrow$, $\Rightarrow$, and $\leadsto$ Where $A \cup \{\phi\}$ is a set of formulas, $A{\rightarrow}\phi$ is a *strict rule*, $A{\Rightarrow}\phi$ is a *defeasible rule*, and $A{\leadsto}\phi$ is an *undercutting defeater*. In each case, we call A the *antecedent* of the rule and we call ϕ the *consequent* of the rule. Where $A = \{\psi\}$, we denote $A{\rightarrow}\phi$ as $\psi{\rightarrow}\phi$, and similarly for defeasible rules and defeaters. Antecedents for strict rules and defeaters must be non-empty; antecedents for defeasible rules may be empty. We will call a rule of the form $\emptyset{\Rightarrow}\phi$ a *presumption* and represent it more simply as ${\Rightarrow}\phi$. While we allow variables in rules, we will treat such rules as schemata for all their instantiations.

Strict rules can never be defeated. They not only do not have exceptions, but could not have exceptions. We may think of them as expressing a necessary connection between antecedent and consequent. Examples of the kind of claim we would represent by a strict rule are “Bachelors are not married” and “Penguins are birds.” Defeasible rules represent weaker connections which can be defeated. Examples are “Penguins live in Antarctica” and “Birds fly.” An example of a presumption is “Presumably, there is no life on the moon.” Undercutting defeaters are too weak to support an inference. Their role is to call into question an inference we might otherwise be inclined to make. We represent such caveats using “might” as in “A damp match might not burn.”

A defeasible theory includes an initial set of facts and a set of rules, but it must include more. Two rules with consequents ϕ and $\neg\phi$ conflict with each other and might defeat each other. Schurtz [9] suggests that this principle should be extended to any rules with incompatible consequents. Makinson suggests that we should detach the consequents of norms (that is, normative defeasible rules) iteratively “while controlling for ‘consistency with the condition’ in a piecemeal way” ([10], page 20.) But recognizing exactly when the consequents of a set of rules will lead to inconsistency is a serious problem, particularly for first-order logic where the question is not decidable. We will address this problem by explicitly incorporating the notion of a conflict set in our defeasible theories. Each conflict set will represent a minimal set of incompatible formulas, and a set of rules conflict if their consequents comprise a conflict set. We want the set of conflict sets in a theory to reflect the necessary relations embodied in our strict rules. If $\{\phi, \psi\}{\rightarrow}\chi$ is in our theory, then $\{\phi, \psi, \neg\chi\}$ should be one of our conflict sets. This idea will be extended below so that our conflict sets are “closed” under the strict rules of a theory.

Another component of a defeasible theory will be a precedence relation. This relation provides a way of adjudicating conflicts between conflicting rules. When we want to apply a defeasible rule $A{\Rightarrow}\phi$, we must look at all conflict sets to which ϕ belongs. In each such set, there must be one member ψ different from ϕ such that for every rule with consequent ψ either the antecedent of the rule fails or $A \Rightarrow \phi$ takes precedence over the rule. More will be said about the precedence relation after the definition of a defeasible proof has been presented.

The theories we will consider, then, consist each of a set of literals (representing initial facts about the world,) a set of rules, a set of conflict sets (closed under the strict rules in the theory,) and a precedence relation.

Definition 1. *A* **closed defeasible theory** *is a quadruple $\langle F, R, C, \prec \rangle$ such that*

1. *F is a set of formulas,*
2. *R is a set of rules,*
3. *C is a set of finite sets of formulas such that for every formula ϕ,*
   (a) $\{\phi, \neg\phi\} \in C$, and
   (b) for every $S \in C$ and $A \rightarrow \phi$ in R, if $\phi \in S$, then $A \cup (S - \{\phi\}) \in C$, and
4. *$\prec$ is an acyclic binary relation on the non-strict rules in R.*

3 Semantics

The most common approach to developing a semantics for nonmonotonic systems is the *fixed-point* approach. In fixed-point systems, we consider supersets of a theory which satisfy certain constraints. These fixed-points are also called extensions of the theory. Typically, every default rule in the system is either failed (the antecedent is not contained in the extension,) defeated, or applied (its consequent is in the extension.) An extension is in some sense a *smallest* set that satisfies this requirement. The sense of 'smallest' used here is not always simply the set theoretic notion. A single theory may have multiple extensions. When a theory has multiple extensions, one option is to take the consequences of the theory to be the intersection of all the extensions of the theory, the so-called "skeptical" approach. In general, either there are no algorithms for generating the extensions of a theory or generating the extensions is computationally expensive. (However, see [11] for some work on testing to see if a formula is a member of an"admissible set" without having to generate the entire set.) Furthermore, a particular theory may not have an extension. Examples of what I am calling fixed-point theories include default logic ([12]), autoepistemic logic ([1,13]), or in the deontic arena, allowed entailments ([14]).

The fixed-point semantics presented here for our defeasible language is due to Donnelly ([15].) First, we want a set of literals that includes all the initial literals in a theory and that also *complies with* all the rules in that theory.

Definition 2. *Let T be a closed defeasible theory. A set K of literals is* **T-compliant** *if and only if*

1. *$F_T \subseteq K$,*
2. *$\phi \in K$ if there is $A \rightarrow \phi \in R_T$ such that $A \subseteq K$, and*

3. *$\phi \in K$ if there is $A \Rightarrow \phi \in R_T$ such that $A \subseteq K$, and for all $S \in C_T$, if $\phi \in S$ then there is $\psi \in S - (F_T \cup \phi)$ such that*
   (a) *for all $B \rightarrow \psi \in R_T$, $B \not\subseteq K$,*
   (b) *for all $B \Rightarrow \psi \in R_T$, either $B \not\subseteq K$ or $B \Rightarrow \psi \prec_T A \Rightarrow \phi$, and*
   (c) *for all $B \rightsquigarrow \psi \in R_T$, either $B \not\subseteq K$ or $B \rightsquigarrow \psi \prec_T A \Rightarrow \phi$.*

Compliance with the rules in a theory is not enough. If it were, we could take our extensions to be those T-compliant sets having no proper subsets that are T-compliant. The problem with this approach is that we can still have gratuitous beliefs represented in such a set. Take for example a theory T for which $K_T = \{\phi\}$, $R_T = \{\phi \Rightarrow \psi, \chi \Rightarrow \theta, \theta \Rightarrow \chi, \{\chi, \theta\} \Rightarrow \sim\psi\}$, $\prec_T = \emptyset$, and C_T just contains all pairs of atomic formulas and their negations. Then intuitively we only want ϕ and ψ in our extensions of T. But the set $S = \{\phi, \chi, \theta\}$ is also a smallest T-compliant set. We would say that from the point of view of T, belief in χ and θ is gratuitous. But each supports the other once they are accepted, and together they defeat the inference to ψ. However, if we remove both χ and θ from S, nothing in the remaining set $\{\phi\}$ together with the rules in R_T requires us to put either of χ and θ back into the set. That is, we can throw these two literals away and what is left does not force us to put them back. This should not happen with an extension of a defeasible theory.

Definition 3. *Let T be a closed defeasible theory. A set of literals E is a* **T-extension** *if and only if*

1. *E is T-compliant, and*
2. *there is no $K \subseteq E$ with $K \neq \emptyset$ such that*
   (a) *$F_T \subseteq E - K$,*
   (b) *$\phi \notin K$ if there is $A \rightarrow \phi \in R_T$ such that $A \subseteq E - K$,*
   (c) *$\phi \notin K$ if there is $A \Rightarrow \phi \in R_T$ such that $A \subseteq E - K$, and for all $S \in C_T$, if $\phi \in S$ then there is $\psi \in (S - (F_T \cup \{\phi\})$ such that*
      i. *for all $B \rightarrow \psi \in R_T$, $B \not\subseteq E - K$,*
      ii. *for all $B \Rightarrow \psi \in R_T$, either $B \not\subseteq E - K$ or $B \Rightarrow \psi \prec_T A \Rightarrow \phi$, and*
      iii. *for all $B \rightsquigarrow \psi \in R_T$, either $B \not\subseteq E - K$ or $B \rightsquigarrow \psi \prec_T A \Rightarrow \phi$.*

We will say that a literal ϕ is *defeasibly entailed* by a defeasible theory T just in case ϕ is in every T-extension.

Definition 4. *Where T is a closed defeasible theory and ϕ is a literal, $T \mathrel{|}\!\approx \phi$ if and only if for every T-extension E, $\phi \in E$.*

4 Proof Theory

As was mentioned before, several well-known approaches to nonmonotonic reasoning develop a fixed-point semantics but provide no proof theory to go with it. In at least some cases, it appears that no constructive proof theory is possible. In the case of defeasible logic, the proof theory came first and the semantics came much later. While a fixed-point semantics without a proof theory might be intellectually satisfying in some respects, practical application is difficult or impossible. And it certainly isn't an attractive model for the nonmonotonic reasoning of ordinary people.[1] After we present the proof theory we will investigate the relationship between the proof theory and the semantics.

The defeasible logic presented here is a refinement of the system presented in [16]. Our proof theory will provide a constructive way to establish that a particular formula is derivable from a theory without having to generate a fixed-point or extension for the theory. Every theory will have a unique closure in the logic.

To apply a defeasible rule, it will sometimes be necessary to show that a conflicting rule is not satisfied, that is, that it's antecedent conditions are not derivable. Thus, a proof will include both positive and negative defeasible assertions.

Definition 5. *σ is a* **positive defeasible assertion** *iff there is a defeasible theory T and a formula ϕ such that $\sigma = T \mid\!\sim \phi$. σ is a* **negative defeasible assertion** *iff there is a defeasible theory T and a formula ϕ such that $\sigma = T \sim\!\mid \phi$. σ is a* **defeasible assertion** *iff σ is either a positive defeasible assertion or a negative defeasible assertion.*

A negative assertion $T \sim\!\mid \phi$ is intended to make a stronger statement than $T \not\mid\!\sim \phi$. $T \sim\!\mid \phi$ indicates that there is a demonstration that ϕ does not follow from T. Before we can detach the consequent of a defeasible rule, we will need to establish that it is not defeated by some conflicting rule. To do this, we will often need to show that the antecedent of a conflicting rule cannot be satisfied. What we need, then, are complementary notions of derivation and refutation. We will sometimes need to show that some formula is refutable in order to show that another formula is derivable, and we will also sometimes need to show that some formula is derivable in order to show that another formula is refutable. Of course, by "refutable" we do not mean that a formula can be shown to be false. Instead, we mean only that we can show that it is not derivable. I think both of these uses of the terms "refute" and "refutation" occur in ordinary usage, but it is important to keep in mind which is intended here.

Our defeasible proofs will have a tree-structure rather than the usual linear structure. Those familiar with logic programming will know that a query can succeed, fail finitely, or fail infinitely. A query fails infinitely when an attempt to prove it can proceed indefinitely without succeeding and without failing in the usual sense. One way a query can fail infinitely is when there is circularity in a theory and the same new query to be proved occurs repeatedly. For example, if our theory contains no formulas and only the single rule $\phi \rightarrow \phi$, then the use of the rule $\phi \rightarrow \phi$ in an attempt to find a proof can lead us to try to prove ϕ by

[1] Of course, we should want our formal systems to *resemble* the reasoning system of ordinary people, but we generally don't want a formal system that mirrors *exactly* what ordinary people do. After all, ordinary people often reason very badly. What we want to do is to discover those patterns where their reasoning is at its best and then try to extend those patterns to correct other cases where their reasoning goes astray.

proving ϕ *ad infinitum*. This is what an automated theorem prover will do if it has no way to check for loops; but of course a proof theory is not committed to any particular method for generating proofs. All that is necessary for ϕ to follow from T is that there is one successful proof of ϕ from T. However, when showing that T *refutes* ϕ, we will need to show that *all* efforts to prove ϕ from T must fail. One way an attempt can fail is by looping. Giving our proofs a tree structure makes it possible to recognize and use such infinite failures within the proof theory. Alternatively, we might try a labeled logic *a lá* Gabbay [17] where the labels carry the same information as the descendants of a node in a proof tree.

Definition 6. *$\mathcal{T}$ is a* **defeasible argument tree** *iff $\mathcal{T}$ is a finite tree and there is a defeasible theory $th(\mathcal{T})$ such that for every node n in $\mathcal{T}$ there is a formula ϕ such that n is labeled either $th(\mathcal{T}) \mid\!\sim \phi$ or $th(\mathcal{T}) \sim\!\mid \phi$.*

Definition 7. *Let σ be a defeasible argument tree and let n be a node in σ. The* **depth of** *n* **in** *σ* **is** *k ($dp(n, \sigma) = k$) iff n has $n - 1$ ancestors in σ. The* **depth of** *σ* **is** *k ($dp(\sigma) = k$) iff $k = max\{j$: there is a node $m \in \sigma$ such that $dp(m, \sigma) = j\}$.*

So far, we have used the notion of the antecedent of a rule succeeding or failing informally. Before presenting our basic defeasible proof theory, we will say precisely what it means for sets of formulas to succeed or fail at a node in a defeasible argument tree.

Definition 8. *Let $\mathcal{T}$ be a defeasible argument tree, $th(\mathcal{T}) = T$, n be a node in $\mathcal{T}$, and A be a set of formulas.*

1. *A* **succeeds at** *n iff for all $\phi \in A$, n has a child labeled $T \mid\!\sim \phi$.*
2. *A* **fails at** *n iff there is $\phi \in A$ such that n has a child labeled $T \sim\!\mid \phi$.*

Definition 9. *$\mathcal{T}$ is a* **defeasible proof** *(***d-proof***) iff $\mathcal{T}$ is a defeasible argument tree, $th(\mathcal{T}) = T$, and one of the following conditions holds for every node n in $\mathcal{T}$.*

1. *n is labeled $T \mid\!\sim \phi$ and either*
   (a) *$\phi \in F_T$,*
   (b) [**Strict Detachment**] *there is $A \rightarrow \phi \in R_T$ such that A succeeds at n, or*
   (c) [**Factual Detachment**] *there is $A \Rightarrow \phi \in R_T$ such that A succeeds at n and for all $S \in C_T$, if $\phi \in S$ then there is $\psi \in S - (F_T \cup \{\phi\})$ such that*
      i. *for all $B \rightarrow \psi \in R_T$, B fails at n,*
      ii. *for all $B \Rightarrow \psi \in R_T$, either B fails at n or $B \Rightarrow \psi \prec_T A \Rightarrow \phi$, and*
      iii. *for all $B \rightsquigarrow \psi \in R_T$, either B fails at n or $B \rightsquigarrow \psi \prec_T A \Rightarrow \phi$.*
2. *n is labeled $T \sim\!\mid \phi$ and there is $S \in C_T$ such that*
   (a) *$\phi \notin F_T$,*
   (b) [**Failure of Strict Detachment**] *for all $A \rightarrow \phi \in R_T$, A fails at n, and*

  (c) **[Failure of Defeasible Detachment]** *for all* $A \Rightarrow \phi \in R_T$*, either*
   i. *A fails at n, or*
   ii. *there is $S \in C_T$ such that $\phi \in S$ and for all $\psi \in S - (F_T \cup \{\phi\})$, either*
     A. *there is $B \rightarrow \psi \in R_T$ such that B succeeds at n,*
     B. *there is $B \Rightarrow \psi \in R_T$ such that B succeeds at n and $B \Rightarrow \psi \not\prec_T A \Rightarrow \phi$, or*
     C. *there is $B \rightsquigarrow \psi \in R_T$ such that B succeeds at n and $B \rightsquigarrow \psi \not\prec_T A \Rightarrow \phi$.*

3. **[Failure by Looping]** *n is labeled $T \sim\!\mid \phi$, n has an ancestor m in $\mathcal{T}$ such that m is labeled $T \sim\!\mid \phi$, and every node in $\mathcal{T}$ between n and m is labeled with a negative defeasible assertion.*

The Failure by Looping condition in Definition 9 requires justification. To apply a defeasible rule, we sometimes need to show that a conflicting rule is not satisfied. We do this by showing that every attempt to derive the antecedent of the conflicting rule fails. Suppose the antecedent contains ϕ. Then we must explore every way that we might derive ϕ. If we discover one way of deriving ϕ that requires us to derive ϕ, then we can be sure that if ϕ is to be derived at all then there must be some way of doing it that is not circular. Thus, we can reject any circular attempts to establish ϕ. This in no way allows us to avoid examining all the non-circular ways that ϕ might be established. What Failure by Looping amounts to is the recognition that if there is no non-circular way to establish a formula, then there is *no* way to establish it.

Definition 10. *ϕ* **is defeasibly derivable from** *T ($T \mid\!\sim_D \phi$) iff there is a d-proof $\mathcal{T}_\phi$ such that the top node in $\mathcal{T}_\phi$ is labeled $T \mid\!\sim \phi$.*

Definition 11. *ϕ* **is defeasibly refutable in** *T ($T \sim\!\mid_D \phi$) iff there is a d-proof $\mathcal{T}_\phi$ such that the top node in $\mathcal{T}_\phi$ is labeled $T \sim\!\mid \phi$.*

For $\mathcal{T}_\phi$ as described in Definition 10 or 11, we will say that $\mathcal{T}_\phi$ *establishes* $T \mid\!\sim_D \phi$ or $T \sim\!\mid_D \phi$ respectively.

Definition 12. *A set of formulas A* **is defeasibly derivable from** *T ($T \mid\!\sim_D A$) iff for all $\phi \in A$, $T \mid\!\sim_D \phi$.*

Definition 13. *A set of formulas A* **is defeasibly refutable in** *T ($T \sim\!\mid_D A$) iff there is $\phi \in A$ such that $T \sim\!\mid_D \phi$.*

Given our informal notion of refutation, it should not be possible to both derive and refute the same formula from a defeasible theory. After all, a refutation is supposed to be a demonstration that the refuted formula *cannot* be derived. The following theorem establishes this essential property for our basic defeasible logic.

Theorem 1. **[Coherence]** *If $T \sim\!\mid_D \phi$, then $T \not\mid\!\sim_D \phi$.*

Another important property of our proof theory is that defeasible rules cannot produce any new contradictions. Any contradictions derivable in the theory depend only on the strict rules and the initial set of literals in the theory.

Definition 14. *$T \vdash_D \phi$ iff $\langle F_T, \{A \rightarrow \psi : A \rightarrow \psi \in R_T\}, C_T, \prec_T \rangle \mid\!\sim_D \phi$.*

Theorem 2. [Consistency] *If $S \in C_T$ and for all $\phi \in S$, $T \mid\!\sim_D \phi$, then for all $\phi \in S$, $T \vdash_D \phi$.*

Now we return to the relationship between our proof theory and our semantics. Donnelly [15] proves the following result.

Theorem 3. [Soundness] *If $T \mid\!\sim_\Sigma \phi$, then $T \mid\!\approx \phi$.*

Donnelly also shows that our proof theory is *not* complete with respect to our semantics. Makinson and Schlecta [18] introduce the notion of a “floating conclusion”. This is a formula that belongs to the intersection of all extensions of a theory even though no rule supporting the formula is satisfied in every extension. Floating conclusions are possible in our semantics. But since our proof theory is sound, we will be unable to derive any floatin conclusions. So long as a theory has multiple extensions, there will be the possibility of floating conclusions. This implies a more general conclusion, not just the conclusion that the defeasible logic presented here is not complete. Given *any* semantics for nonmonotonic reasoning and *any* proof theory for that semantics, if floating conclusions are possible in that semantics then the proof theory cannot be both sound and complete.

5 Priorities, Specificity, and Normative Reasoning

One method for determining priorities of rules in defeasible theories is to use *specificity.* A rule with antecedent A is said to be more specific than a rule with antecedent B, relative to a theory T, if we can derive all of B from A using only the rules in T, but not *vice versa.* The precedence relation in a theory might be based on such a notion of specificity. A more interesting case, though, is where we already have some explicit relation on the rules of a theory that we want to use as the *core* of our precedence relation. We might then extend this core precedence relation by using specificity in all those cases where the core precedence relation does not settle the matter. To state this condition precisely, we will let $R^\circ = \{r : r \in R$ and the antecedent of r is not empty $\}$.

Definition 15. *Let $\Gamma \subseteq \prec_T$. Then T is* **Γ-specific** *iff for all non-strict $r_1, r_2 \in R_T$ such that $(r_1, r_2) \notin \Gamma$, A is the antecedent of r_1, and B is the antecedent of r_2,*

1. *if $\langle A, R_T^\circ, C_T, \prec_T \rangle \mid\!\sim_D B$ and $\langle B, R_T^\circ, C_T, \prec_T \rangle \sim\!\mid_D A$, then $r_2 \prec_T r_1$, and*
2. *if $\langle A, R_T^\circ, C_T, \prec_T \rangle \sim\!\mid_D B$ or $\langle B, R_T^\circ, C_T, \prec_T \rangle \mid\!\sim_D A$, then $r_2 \not\prec_T r_1$.*

A special case will be where we use specificity as our sole criterion for adjudicating conflicts between non-strict rules. This amounts to taking the empty set as our core precedence relation.

Definition 16. *A theory T* **preserves specificity** *iff T is $\emptyset$-specific.*

An advantage of using Γ-specific or specificity preserving theories is that we can now *compute* whether one rule takes precedence over another. This becomes particularly important when we add a deontic operator O (for 'ought') to our logic and represent norms (rules with deontic consequents.) The proof theory for deontic defeasible logic requires additional principles to handle the interaction between norms and other rules and to resolve some of the paradoxes in standard deontic logics. Such a deontic extension of defeasible logic is described in [16,19].

Let's call a theory T that contains norms a *primary* theory, and let's call a theory $T^{\prec}$ that contains rules about which rules in T take precedence over which other rules in T a *precedence* theory for T. By doing proofs or refutations in $T^{\prec}$, we can determine in at least some cases whether $A{\Rightarrow}\phi \prec_T B{\Rightarrow}\psi$. Specificity is one way to assign priorities for rules, but there are others. The principles of *lex superior* and *lex posterior* are familiar examples. Suppose we have a theory T containing rules representing both federal laws of the United States and laws of some particular state within the United States. In the language of a second theory $T^{\prec}$, we have names for all the rules in T and predicates **federal**, **state** and $\sqsubset$. Then $T^{\prec}$ can contain all instances of the form

$$\{\mathbf{federal}(r_1), \mathbf{state}(r_2)\}{\Rightarrow}r_2 \sqsubset r_1.$$

This would be a special instance of the principle of *lex superior*. Let

$$\Gamma = \{\langle r_1, r_2\rangle : T^{\prec} \mid\!\sim_D r_1 \sqsubset r_2\}$$

and suppose T is Γ-specific. Now we can use a combination of *lex superior* and specificity to determine precedence among the laws in our theory T of normative use. And we can use proofs and refutations in $T^{\prec}$ to determine the core of the precedence relation for T. Our complete defeasible theory now has two components, a primary theory and a theory of precedence for the primary theory. Notice that *lex superior* takes priority (at a higher level) over specificity since the definition of Γ-specificity guarantees that specificity is only applied when the core precedence relation Γ does not determine priority.

Clearly, matters become more complex when we try to add *lex posterior* to our precedence theory and when we try to include the laws of all fifty states in the U.S. It may even turn out to be impossible to do this without violating the requirement that $\prec_T$ must be non-cyclical. It is an open question whether for any non-cyclic relation Γ on the non-strict rules in a set R of rules, any set F of facts, and any conflicts set C for the language of F and R, it is possible to construct a theory $T = \langle F, R, \prec, C\rangle$ that is Γ-specific.

Notice in the example of *lex superior*, the rule used to express this principle is defeasible. But why shouldn't this be a strict rule? Indeed, why should we need defeasibility at all in a precedence theory? The doctrine of states' rights found in the U.S. Constitution illustrates the need for defeasible rules in precedence theories. Once again, *lex superior* should tell us that Constitutional law should take precedence over either federal or state law. So we get a nice, neat three-level normative system. But the principle of states' rights is that the Constitution prohibits the federal government from enacting laws limiting the powers of the state governments in certain areas. If this principle of states' rights were to prevail, then the principle of *lex superior* would have to be defeasible. It would have exceptions in any case where a federal law contradicted a state law and the subjects of these two conflicting laws fell under the umbrella of the Constitutional guarantee of states' rights.

6 Implementation

6.1 d-Prolog

The design of defeasible logic has been influenced by issues of implementation, and particularly of implementation in Prolog, from the beginning. d-Prolog is a nonmonotonic extension of the Prolog programming language. There have been several versions of d-Prolog dating back to 1986 (see [20].) The latest version is an implementation of an earlier version of defeasible logic described in [21,22]. A complete description of this version of d-Prolog is in [23, chapter 11].

Here are some of the features of the latest version of d-Prolog, including some of the ways the defeasible logic it implements differs from the formal logic described earlier in this paper.

1. d-Prolog does not implement Failure by Looping.
2. d-Prolog is *not* closed under strict rules. Conflict pairs have only two members. All pairs of literals and their complements form a conflict pair. Other conflict pairs can be listed explicitly using the predicate `incompatible/2`.
3. Because d-Prolog is not closed under strict rules, inference using strict rules is handled differently. When the conditions of two conflicting strict rules are only defeasibly satisfied, d-Prolog treats the two rules as though they defeat each other and the consequent of neither rule is detached. The purpose of this strategy is to limit the contradictions that can be derived to those that follow from the strict rules and initial facts alone.
4. In d-Prolog, the query

   ```
   ?- spec.
   ```

   toggles between enabling and disabling specificity. In response to the query, d-Prolog informs the user whether specificity has just been enabled or disabled. Even if specificity is enabled, the programmer can still add clauses for the predicate `sup/2` to force resolution of conflicts between rules where specificity fails to determine superiority. When d-Prolog is loaded, specificity is enabled by default.

We begin by defining the language of d-Prolog. One unary functor `neg` and two binary infix functors `:=` and `:^` are added to Prolog. `neg` is a sound negation operator which we distinguish from the built-in negation-by-failure (NBF) operator `not` or `\+`. Where `Atom` is an atomic clause, `Atom` and `neg Atom` are *complements* of each other. `neg Atom` can occur in either the Head or the Body of a rule. Clauses of the form `Head := Body` are called *defeasible* rules, and clauses of the form `Head :^ Body` are called *undercutting defeaters* or simply *defeaters*. A defeasible rule of the form `Head := true` is called a *presumption*. By contrast, ordinary Prolog rules are called *strict* rules.

Conclusions may be derivable either strictly or defeasibly. A conclusion is derivable strictly if and only if it is derivable from the facts and strict rules in the knowledge base alone. A clause `Goal` is strictly derivable, then, just in case the query `?- Goal.` succeeds. A conclusion is defeasibly derivable if it is derivable using all the clauses in the knowledge base including defeasible rules, presumptions, and defeaters. Thus, a conclusion is defeasibly derivable if it is strictly derivable. We introduce a new unary functor `@` to invoke the defeasible inference engine, and a clause `Goal` is defeasibly derivable just in case the query `?- @ Goal.` succeeds. We read `@ Goal` as 'Defeasibly, Goal' or as 'Apparently, Goal'.

Another unary functor `@@` is introduced to support exhaustive investigation of queries. We want a way to find out with a single query whether a *ground* atomic clause or its negation is either strictly or defeasibly derivable. In response to the query `?- @@ Goal`, d-Prolog will test for all these possibilities and give an appropriate report: 'definitely yes', 'definitely no', 'presumably yes', 'presumably no', or 'can't tell'.

Here is an example of a d-Prolog knowledge base.

```
born_in(X,usa) :- born_in(X,atlanta).
neg born_in(X,usa) :=
   native_speaker(X,greek).
born_in(stavros,atlanta) := true.
native_speaker(stavros,greek) := true.
```

For this knowledge base, d-Prolog responds to the query `?- @@born_in(stavros, usa).` with the report 'presumably yes' because the strict rule is superior to the defeasible rule.

Another familiar example, the so-called Nixon Diamond, demonstrates how two defeasible rules may defeat each other. We represent this example in d-Prolog by the following facts and rules.

```
pacifist(X) := quaker(X).
neg pacifist(X) := republican(X).
quaker(nixon).
republican(nixon).
```

The correct response to the query

```
?- @@ pacifist(nixon)
```

is 'can't tell' since neither of the two defeasible rules is superior to the other and each is rebutted by the other. However, we could decide that in this kind of situation political party takes priority over religious affiliation. Then we can add the following clause to our d-Prolog knowledge base.

```
sup((pacifist(X) := quaker(X)),
  (neg pacifist(X) := republican(X))).
```

With this addition, the query

```
?- @@ pacifist(nixon).
```

produces the report 'presumably no'.

We have begun work on a new version of d-Prolog that implements conflict sets closed under strict rules and Failure by Looping.

6.2 d-GRAPHER

Knowledge based systems (KBSs) that model inference about specific domains incorporate representations of the knowledge necessary to solve problems in their domains. Another kind of decision support tool is needed that allows users to model knowledge not already represented in the system. Such an *argumentation based system* (ABS) would provide tools to help the user represent knowledge about *any* domain. It would incorporate an inference mechanism to help the user derive conclusions from the knowledge that has been modeled. The system would make the inference process visible to the user and allow the user to construct a variety of "what-if" scenarios easily and quickly. While a KBS applies preselected argument structures to the information provided by the user, an ABS would allow the user to construct and evaluate competing arguments on any subject before making a decision. Systems of this sort have been reported in [24,25].

d-GRAPHER ([26] is an argumentation based system built around an implementation of defeasible logic. Knowledge is represented in d-GRAPHER using defeasible graphs, graphical representations of defeasible logic theories. Nodes in a defeasible graph represent atoms (corresponding to consequents of rules) or sets of literals (corresponding to the antecedents of rules.) Nodes are connected by six kinds of arrows representing positive or negative strict, defeasible, or defeater links. A negative link is a $\rightarrow$, a $\Rightarrow$, or a $\leadsto$ with a bar through it. A negative link such as $A \not\rightarrow \phi$ represents the rule $A \rightarrow \sim\phi$. A literal in a d-graph is marked true by displaying it in green and marked false by displaying it in red. If it can be shown that neither an atom nor its negation can be derived, then both are displayed in yellow. In [27] we describe an algorithm for propagating markings (colors) in a defeasible graph, and we showed that for finite acyclical defeasible graphs with an initial partial marking, this algorithm eventually marks every node in the graph. We also showed that this algorithm is sound with respect to the earlier version of defeasible logic implemented in d-Prolog. This means that our graph marking algorithm provides a sound, decidable proof theory for the fragment of defeasible logic represented by defeasible graphs.

d-GRAPHER is a tool for building and reasoning with defeasible graphs. The d-GRAPHER interface, written in Visual Basic, allows the user to build and label defeasible graphs using a drag-and-drop method. The user can then mark an initial set of atoms in the graph green (true) or red (false). When the user clicks a button to begin inference, a representation of the graph is sent to a

Prolog inference engine. A complete set of markings is returned to the interface and the graph is marked accordingly.

7 Applications

7.1 An Expert System

At least one expert or decision support system has been developed that uses d-Prolog. FORE ([28]) helps a user select a business forecasting method from among twenty available methods. The selection is based on sixteen different criteria. An analysis of these criteria and how they affect the choice of a forecasting method appeared in [29]. The results are organized into a matrix. Within each cell of the matrix is an explanation of how the corresponding criterion strongly or weakly indicates or counter indicates the corresponding method. These explanations were converted into Prolog rules. A small core of d-Prolog rules were then constructed to control the way in which these indicators and counter indicators would determine the final selection of a forecasting method. Using d-Prolog to control the way the other rules interacted made it much easier to write the large set of rules representing the selection matrix. In particular, it is not necessary to write exceptions into each rule. d-Prolog allows exceptions to be added as new rules without altering rules that are already in the knowledge base.

The FORE interface asks the user questions about the user's forecasting requirements, data, and other criteria affecting the selection of a forecasting method. These questions are asked on a need-to-know basis meaning that answers to earlier questions can determine whether or not a question needs to be asked later. The answers are used to determine which methods are strongly or weakly indicated or counter indicated. Then the core of d-Prolog rules is used to arrive at overall recommendations, determining which methods are preferred given the user's situation. The system then makes its recommendations and provides an explanation for its recommendations.

7.2 Planning and Learning

Defeasible logic looks promising for use in planning and machine learning, and some initial work has been done in these areas ([30].) This work involves efforts to develop agents that can perform complex tasks in an artificial environment. The environment, V-World, is a simulator written in Prolog ([31].) It supports development of artificial worlds with complex ontologies similar to those seen in some video games. Agents can then be developed that interact with these worlds, learning the ontology of the world through interaction with objects and other actors and/or working to achieve a goal that requires complex planning. A graphical interface allows the user to observe the actions of 'softbots' as they move about a virtual world.

When an agent interacts with an object or actor in a V-World environment, the agent's strength or damage level may increase or decrease as a consequence of the interaction. In our learning experiments, the agent noted a most salient feature of an object with which it interacted and combined this feature with the consequences of the action in a defeasible rule. For example, if the agent pushed against an object whose most salient feature was that it was a tree and the interaction resulted in damage, then the agent might learn the rule

```
damage_incurred :=
      tree(X),
      push (X).
```

This rule would guide the agent to avoid trees when its damage level was high; but the imperative to experiment built into the agent would prompt the agent to interact with a tree on another occasion if it was strong and undamaged. On the next occasion, the agent might experience no damage and an increase in strength when it pushed a tree. It would then look for a next most salient feature of the tree it had just encountered and learn a new, more specific rule such as

```
neg damage_incurred :~
      tree(X),
      fruit_bearing(X),
      push(X).

strength_enhanced :=
      tree(X),
      fruit_bearing(X),
      push(X).
```

A third encounter with a tree might result in a rule like

```
damage_incurred :=
      tree(X),
      fruit_bearing(X),
      thorny(X),
      push(X).
```

Of course, it might have been thorns that caused the damage in the original encounter, but since we are assuming that this was not the most salient feature of the tree, it was not the feature that the agent associated with the damage. Each time the agent experiments with a kind of entity that it has previously experienced, it uses the rules it has learned to predict the outcome of the experiment. When the outcome is different from the prediction, the agent determines the rule that supported the prediction, determines some feature of the entity that is not included in the rule, forms a new rule that will overrule or at least defeat the original rule, and adds it to its store of knowledge. We found that eventually an agent developed rules that allowed it to predict the outcomes of experiments unerringly in the relatively simple environments of V-World. In the process, some of the simpler rules learned early in the agent's history became useless appendages that were always defeated by more specific rules learned later. If we added a procedure that eventually removed rules that had not 'fired' for a long time, we might have a reasonable though primitive model for how an agent learns the ontology of its environment and of how that ontology is represented.

Just as the rules learned by our learning agent were defeasible, so were the plans constructed by our planning agent. In these experiments, the agent began with a full understanding of the ontology of its environment. Its goal was to learn the geography of its environment, to locate some object, and to transport this object to some location. Its task was complicated by various obstacles: doors that required keys, guardians that could cause damage and could only be safely by-passed through successful attacks or bribes, etc. The agent began with a simple plan represented in a presumption, for example,

```
valid_plan([find(gold), take(gold), goto(castle)]) :=
      true.
```

If the agent found the gold and discovered that the gold was guarded by a dragon, and it knew that a sword was required to eliminate dragons, then it would add a rule like the following to its knowledge base.

```
valid_plan([find(sword), goto(dragon), attack(dragon),
   goto(gold), take(gold), goto(castle)] :=
      true.
```

On the other hand, if the agent located the gold and it was unguarded, then it would *remove* the rule holding the plan it had just acted on, delete the first step, and assert the new, simpler rule:

```
valid_plan([take(gold), goto(castle)]) :-
      true.
```

The agent also began with a set of rules telling it when plans were not valid. Here are some examples.

```
neg valid_plan([Step|_]) :=
      strength(low),
      Step \== goto(food).

neg valid_plan([Step|_]) :=
      Step = take(Object),
      guarded_by(Object, Actor).
```

The agent would also have a library of rules for very simple plans for certain situations, for example,

```
valid_plan([goto(food)]) :=
      strength(low).
```

Each move, the agent determined whether it could derive that there was a valid plan according to its rules. If it derived a valid plan, then it based its next move on that plan. If it could not derive a valid plan, then it called its planning routine to generate a new plan that would be valid for at least one move.

This approach to planning had some interesting consequences. Plans that became invalid were not deleted. Plans were only deleted, one step at a time, as they were executed. Thus at the end of a session it was possible to examine plans that the agent had developed along the way but at some point abandoned. But it was also possible for a plan to become invalid and then later to become valid again. This would allow the agent to return to a revalidated plan and pursue it again without having to develop a new plan.

Only a small number of experiments to investigate the value of defeasible logic in learning and planning in simple environments have been completed, but these experiments show promise for the approach. I hope to return to this line of research in the near future.

7.3 Normative Reasoning

In an earlier section I alluded to an extension of defeasible logic that incorporated deontic operators. Hunter developed an extension of d-Prolog that implemented some of the features of this defeasible deontic logic in [32]. But prior to that, other researchers had developed machinery for performing normative reasoning within the framework of d-Prolog.

Ryu ([33]) used d-Prolog to model the lending policies of the University of Texas and to model small secured loans (pawns) as described in the Uniform Business Code. Dhanesha ([34]) later used d-Prolog to model the parking regulations of George Mason University. Each of these authors built additional machinery on top of the d-Prolog foundation to handle specific issues that arose with regard to normative reasoning, although the way the two authors approached these problems were rather different. In any case, both of these studies have an *ad hoc* flavor and give the impression that the models were developed pragmatically and with limited consideration for the underlying theoretical issues involved in applying defeasible reasoning to normative contexts. Ryu and Lee published several later papers that explored the theoretical questions ([35,36,37].)

8 Conclusions

The relative amount of effort that has been expended on theory development, implementation, and applications for defeasible logic seems to me to mirror the situation for research on nonmonotonic formalisms generally. A great deal of work has gone into the development of theory and this work continues in part because convergence has not occurred. Considerably less work has been done on implementing nonmonotonic reasoning systems and, as with defeasible logic, this work tends to lag behind the theoretical development. In some cases, and this is certainly true for defeasible logic, experience with implementations of earlier systems has uncovered issues that have led to refinement of the underlying theories. Finally, the number of applications involving nonmonotonic reasoning systems remains quite small when compared to the effort that has gone into theoretical work. But applications will not appear until implementations of non-monotonic inference systems become more widely available, and confidence in both the theoretical work and implementations based on it depends finally on the development of interesting applications.

References

1. Moore, R.: Semantical considerations on non-monotonic logic. Artificial Intelligence **25** (1985) 75–94
2. Loui, R.: Theory and Computation of Uncertain Inference and Decision. PhD thesis, The University of Rochester (1987) Technical Report 228, Department of Computer Science.
3. Loui, R.: Defeat among arguments: A system of defeasible inference. Computational Intelligence **3** (1987) 100–106
4. Geffner, H.: Default Reasoning: Causal and Conditional Theories. PhD thesis, UCLA (1989) Research Report 137, Cognitive Systems Laboratory, Department of Computer Science.
5. Geffner, H., Pearl, J.: A framework for reasoning with defaults. In Kyburg, H., Loui, R., Carlson, G., eds.: Knowledge Representation and Defeasible Reasoning. Studies in Cognitive Systems. Kluwer Academic Publishers, Boston (1989) 69–88
6. Pollock, J.: A theory of defeasible reasoning. International Journal of Intelligent Systems **6** (1991) 33–54
7. Pollock, J.: Self-defeating argument. Minds and Machines **1** (1991) 367–392
8. Nute, D.: Defeasible logic. In Gabbay, D., Hogger, C., eds.: Handbook of Logic for Artificial Intelligence and Logic Programming. Volume III. Oxford University Press, Oxford (1994)
9. Schurtz, G.: Defeasible reasoning based on constructive and cumulative rules. In Casati, R., Smith, B., White, G., eds.: Philosophy and Cognitive Sciences. Hölder-Pichler-Tempsky (1994) 297–310
10. Makinson, D.: On a fundamental problem of deontic logic. In Prakken, H., McNamara, P., eds.: ΔEON'98: 4th International Workshop on Deontic Logic in Computer Science, Università degli Studi di Bologna (1998)
11. Dung, P.M., Kowalski, R.A., Toni, F.: Synthesis of proof procedures for default reasoning. In: Proceedings of the international workshop on logic programming synthesis and transformation, Springer Lecture Notes on Computer Science 1207 (1996) 313–324
12. Reiter, R.: A logic for default reasoning. Artificial Intelligence **13** (1980) 81–132
13. Konolige, K.: On the relation between default theories and autoepistemic logic. Artificial Intelligence **35** (1988) 343–382
14. Morreau, M.: Reasons to think and act. In Nute, D., ed.: Defeasible Deontic Logic. Synthese Library. Kluwer Academic Publishers, Dordrecht, Netherlands (1997) 139–158
15. Donnelly, S.: Semantics, soundness, and incompleteness for a defeasible logic. Master's thesis, Artificial Intelligence Center, The University of Georgia (1999)
16. Nute, D.: Apparent obligation. In Nute, D., ed.: Defeasible Deontic Logic. Synthese Library. Kluwer Academic Publishers, Dordrecht, Netherlands (1997) 287–315
17. Gabbay, D.: Labelled Deductive Systems. Volume 1. Oxford University Press, Oxford (1996)

18. Makinson, D., Schechta, K.: Floating conclusions and zombie paths: two deep difficulties in the 'directly skeptical' approach to inheritance nets. Artificial Intelligence **48** (1991) 199–209
19. Nute, D.: Norms, priorities, and defeasibility. In McNamara, P., Prakken, H., eds.: Norms, Logics and Information Systems. IOS Press, Amsterdam (1999) 201–218.
20. Nute, D., Lewis, M.: A users manual for d-Prolog. Research Report 01-0016, Artificial Intelligence Programs, The University of Georgia (1986)
21. Nute, D.: Basic defeasible logic. In nas del Cerro, L.F., Penttonen, M., eds.: Intensional Logics for Programming. Oxford University Press (1992) 125–154
22. Nute, D.: A decidable quantified defeasible logic. In Prawitz, D., Skyrms, B., Westerstahl, D., eds.: Logic, Methodology and Philosophy of Science IX. Elsevier Science B. V, New York (1994) 263–284
23. Covington, M., Nute, D., Vellino, A.: Prolog Programming in Depth, Second Edition. Prentice-Hall, Englewood Cliffs, NJ (1997)
24. Conklin, J., Begemena, M.L.: gIBIS: A tool for all reasons. Journal of the American Society for Information Systems **40** (1989) 140–152
25. Hua, H., Kimbrough, S.: On hypermedia-based argumentation decision support systems. unpublished manuscript (1995)
26. Nute, D., Henderson, C., Hunter, Z.: Defeasible logic graphs ii: Implementation. Decision Support Systems (to appear)
27. Nute, D., Erk, K.: Defeasible logic graphs i: Theory. Decision Support Systems (to appear)
28. Nute, D., Mann, R., Brewer, B.: Conrtolling expert system recommendations with defeasible logic. Decision Support Systems **6** (1990) 153–164
29. Georgoff, D.M., Murdick, R.G.: Manager's guide to forecasting. Harvard Business Review (1986) 110–120
30. Puppa, A.: A comparison: Knowledge representation in Prolog and in defeasible Prolog. Master's thesis, Artificial Intelligence Center, The University of Georgia (1997)
31. Nute, D.: V-World. Software, Artificial Intelligence Center, The University of Georgia (2001) Available online at http:/www.arches.uga.edu/∼dnute/vworld.
32. Hunter, Z.: dd-Prolog: A deontic extension of d-Prolog. Master's thesis, The University of Georgia (1997)
33. Ryu, Y.: A Formal Representation of Normative Systems: A Defeasible Deontic Reasoning Approach. PhD thesis, University of Texas (1992)
34. Dhanesha, K.: Normative expert system using deontic logic and defeasible reasoning. Master's thesis, The University of Georgia (1994)
35. Ryu, Y., Lee, R.: Defeasible deontic reasoning: A logic programming model. In Meyer, J.J.C., Wieringa, R.J., eds.: Deontic Logic in Computer Science: Normative System Specification. John Wiley & Sons Ltd. (1993)
36. Ryu, Y., Lee, R.: Defeasible deontic reasoning and its applications to normative systems. Decision Support Systems **4** (1995) 59–73
37. Ryu, Y., Lee, R.: Deontic logic viewed as defeasible reasoning. In Nute, D., ed.: Defeasible Deontic Logic: Essays in Nonmonotonic Normative Reasoning. Kluwer Academic Publishers, Dordrecht, Holland (1997)