

Propositional Clausal Defeasible Logic

David Billington

School of ICT, Nathan campus, Griffith University,
Brisbane, Queensland 4111, Australia.
`d.billington@griffith.edu.au`

Abstract. Defeasible logics are non-monotonic reasoning systems that have efficient implementations and practical applications. We list several desirable properties and note that each defeasible logic fails to have some of these properties. We define and explain a new defeasible logic, called clausal defeasible logic (CDL), which has all these properties. CDL is easy to implement, consistent, detects loops, terminates, and has a range of deduction algorithms to cater for a range of intuitions.

Keywords: Defeasible logic, Non-monotonic reasoning, Knowledge representation and reasoning, Artificial intelligence.

1 Introduction

Non-monotonic reasoning systems represent and reason with incomplete information where the degree of incompleteness is not quantified. A very simple and natural way to represent such incomplete information is with a defeasible rule of the form “antecedent \Rightarrow consequent”; with the meaning that provided there is no evidence against the consequent, the antecedent is sufficient evidence for concluding the consequent. Creating such rules is made easier for the knowledge engineer as each rule need only be considered in isolation. The interaction between the rules is the concern of the logic designer.

Reiter’s normal defaults [24] have this form, with the meaning that if the antecedent is accepted and the consequent is consistent with our knowledge so far then accept the consequent. Of course the consequent could be consistent with current knowledge and yet there be evidence against the consequent. This results in multiple extensions. However multiple extensions are avoided by interpreting a defeasible rule as “if the antecedent is accepted and all the evidence against the consequent has been nullified then accept the consequent”. This interpretation forms the foundation of a family of non-monotonic logics all based on Nute’s original defeasible logic [22]. (Different formal definitions of “accepted”, “evidence against”, and “nullified” have been used by different defeasible logics.)

Unlike other non-monotonic reasoning systems, these defeasible logics use Nute’s very simple and natural “defeasible arrow” to represent incomplete information. This simplicity and naturalness is important when explaining an implementation to a client. All defeasible logics have a priority relation on rules. Although preferences between rules can be simulated by more complex rules, this is not so natural or simple. Defeasible logics use classical negation rather than

negation-as-failure, and have a type of rule which warns that a conclusion, c , is too risky but does not support the negation of c . Many defeasible logics cater for different intuitions about what should follow from a reasoning situation.

A key feature of these defeasible logics is that they all have efficient easily implementable deduction algorithms (see [8, 20, 23] for details). Indeed defeasible logics have been used in an expert system, for learning and planning [23], in a robotic dog which plays soccer [9, 10], and to improve the accuracy of radio frequency identification [11]. Defeasible logics have been advocated for various applications including modelling regulations and business rules [3], agent negotiations [13], the semantic web [1, 4], modelling agents [16], modelling intentions [15], modelling dynamic resource allocation [17], modelling contracts [12], legal reasoning [18], modelling deadlines [14], and modelling dialogue games [25]. Moreover, defeasible theories, describing policies of business activities, can be mined efficiently from appropriate datasets [19].

The unique features and diverse range of practical applications show that defeasible logics are useful and their language is important for knowledge representation and reasoning. Using defeasible logic as the inference engine in an expert system is obvious. But it is less obvious to use defeasible logic to deal with the error-prone output of sensors (possibly in a robot), because this can be done using classical logic. The advantages of using defeasible logic are that the system can be developed incrementally, there are fewer rules, and the rules are simpler [9–11].

In this paper we shall define a new defeasible logic, called clausal defeasible logic (CDL), explain how it works, and show why it is needed.

The rest of the paper has the following organisation. Section 2 shows why CDL is needed. Section 3 gives an overview of CDL. The formal definitions of CDL are in Sections 4 and 5. An explanation of the proof algorithms concludes Section 5. An example is considered in Section 6. Section 7 contains results that show CDL is well-behaved. A summary forms Section 8.

2 Why Clausal Defeasible Logic is Needed

There are three classes of defeasible logic: the twin logics NDL and ADL [21] and their variants; the logic DL93 [5] and its variants; and plausible logic [8] and its later developments. Table 1 compares CDL and representatives from the three classes by noting which properties hold. From the first class we choose NDL and ADL, from the second class we choose DL93, and from the third class we choose the latest plausible logic PL05 [7].

There are two well-informed but different intuitions about what follows from the defeasible theory $Ambig = \{r_1: \{\} \Rightarrow b, r_2: \{\} \Rightarrow a, r_3: \{\} \Rightarrow \neg a, r_4: \{a\} \Rightarrow \neg b\}$. Rule r_1 says there is evidence for b ; r_2 says there is evidence for a ; r_3 says there is evidence for not a ; and r_4 says that a is evidence for not b . All rules have the same reliability or strength. Because there is equal evidence for and against a , we say a is *ambiguous*. The *ambiguity blocking* intuition says that since a is not accepted, r_4 cannot be used and so b should be accepted because of

Table 1. Logics and their Properties

Property \ Logic	CDL	PL05	NDL	ADL	DL93
Ambiguity blocking	Yes	Yes	Yes	No	Yes
Ambiguity propagating	Yes	Badly	No	Yes	No*
General conflict	Yes	Yes	Yes	Yes	No
Prove disjunctions	Yes	Yes	No	No	No
Team defeat	Yes	Yes	No	No	Yes
Failure-by-looping	Yes	No	Limited	Limited	No
Linear proof hierarchy	Yes	No	Yes	Yes	No*
Terminates	Yes	Yes	No	No	No
Weak decisiveness	Yes	Yes	Yes	No	Yes

* There is a variant of DL93 [2] that has these properties.

r_1 . The *ambiguity propagating* intuition says that although r_4 has been weakened by the ambiguity of a , it has not been weakened enough to ignore r_4 . So r_1 is evidence for b and r_4 is still evidence against b . Thus b should also be ambiguous, that is the ambiguity of a has been propagated along r_4 to b .

Defeasible logics deal with factual, as well as defeasible, information. A defeasible logic has the *general conflict* property iff the logic recognises that two defeasible rules, r_1 and r_2 , conflict whenever the factual information means that the consequents of r_1 and r_2 cannot both hold. For example if $c \vee d$ is a fact then only a logic with the general conflict property will recognise that $\{a\} \Rightarrow \neg c$ and $\{b\} \Rightarrow \neg d$ conflict.

Only plausible logics can *prove disjunctions* of literals.

Suppose there is a team of rules supporting a and a team of rules supporting $\neg a$, and there is a priority between rules. If there is a rule supporting a that is superior to all the rules supporting $\neg a$ then clearly a should be concluded. This is what NDL and ADL do. But more intuitive results can be obtained if the following *team defeat* property is used. If every rule supporting $\neg a$ is inferior to some rule supporting a then conclude a .

To illustrate *failure-by-looping* consider the following example of a positive loop. $\{r_c: \{\} \Rightarrow c, r_{ab}: \{a\} \Rightarrow b, r_{ba}: \{b\} \Rightarrow a, r_{ac}: \{a\} \Rightarrow \neg c\}$. Rule r_c is evidence for c , and r_{ac} is evidence against c . But a cannot be proved because r_{ab} and r_{ba} form a positive loop. Hence the evidence against c has been nullified and so we conclude c . This is what NDL and ADL do.

The following is an example of a negative loop. $NegLoop = \{r_a: \{\} \Rightarrow a, r_b: \{\} \Rightarrow b, r_c: \{\} \Rightarrow c, r_{ab}: \{a\} \Rightarrow \neg b, r_{ba}: \{b\} \Rightarrow \neg a, r_{ac}: \{a\} \Rightarrow \neg c\}$. Again r_c is evidence for c , and r_{ac} is evidence against c . But a cannot be proved because r_{ab} and r_{ba} form a negative loop. Hence the evidence against c has been nullified and so we conclude c . No previous defeasible logic does this.

A logic has a *linear proof hierarchy* iff for any two of its proof algorithms, whatever can be proved by one can be proved by the other or vice versa. For NDL and ADL this means that NDL can prove whatever ADL can prove.

Defeasible logics are designed to be implemented on a computer, and so it would be tidy if their deduction algorithms always *terminated*. The algorithms used in DL93, NDL, and ADL do not terminate when trying to prove c in the *NegLoop* example.

To nullify evidence defeasible logics have to be able to disprove formulas. This means that they can demonstrate in a finite number of steps that there is no proof of the formula. (It does not mean that the negation of the formula can be proved.) It would be nice if every formula could be proved or disproved, called decisiveness in [6]. Unfortunately there are examples that show that decisiveness leads to undesirable results [6]. *Weak decisiveness* means that every formula can be proved, or disproved, or the attempted proof gets into a loop. If such a loop is detected then the proof can be terminated, otherwise the deduction algorithm does not terminate.

For each of the first 6 properties in Table 1, a logic which has this property will give more intuitive results than a logic which does not have the property.

Having a linear proof hierarchy is important because then one does not have to decide which proof algorithm to use. Always use the strongest (most reliable) algorithm. If it succeeds then the weaker algorithms will succeed too. If it fails then use the next weakest (less reliable) algorithm. So different degrees of confidence in a proved result can be achieved without using numbers such as probabilities or certainty factors. Ambiguity propagation gives more reliable results, but proves less, than ambiguity blocking. Moreover a range of ambiguity propagating algorithms can be formed giving a range of reliabilities.

Since CDL is based on PL05 we shall consider the flaws of PL05 more closely. The ambiguity propagating algorithm of PL05 is too strong. Its nullifying sub-algorithm is too weak, which makes PL05 fail to have a linear proof hierarchy. We define two totally new ambiguity propagating algorithms based on ideas exhibited in a variant of DL93 [2].

PL05 detects all loops, but it only uses this information to terminate loops. NDL and ADL do not really detect loops, they just use some loops in their failure-by-looping method. If all loops were used in a failure-by-looping method then the logic would be decisive; which, as noted above, leads to undesirable results. CDL determines which loops are usable and which are not. Every loop used by NDL and ADL is regarded as usable by CDL. Moreover CDL gets the desired answer for the *NegLoop* example, see Section 6.

3 Overview of Clausal Defeasible Logic (CDL)

CDL reasons with both factual and plausible information, which is represented by strict rules, defeasible rules, warning rules, and a priority relation, $>$, on the rules. All rules have the form “finite-set-of-literals arrow literal”.

Strict rules, for example $A \rightarrow l$, represent the aspects of a situation that are certain. If all the literals in A are proved then l can be deduced, no matter what the evidence against l is.

Defeasible rules, for example $A \Rightarrow l$, mean that if all the literals in A are proved and all the evidence against l has been nullified then l can be deduced.

Warning rules, for example $A \rightsquigarrow \neg l$, warn against concluding usually l , but do not support usually $\neg l$. For example, “Sick birds might not fly.” is represented by $\{sick(x), bird(x)\} \rightsquigarrow \neg fly(x)$. The idea is that a bird being sick is not sufficient evidence to conclude that it usually does not fly; it is only evidence against the conclusion that it usually flies.

The priority relation, $>$, on the set of non-strict rules allows the representation of preferences among non-strict rules. The priority relation must be acyclic, but does not have to be transitive. Consider the following two rules.

$$\begin{aligned} r_1: \{bird(x)\} &\Rightarrow fly(x) && \text{[Birds usually fly.]} \\ r_2: \{quail(x)\} &\Rightarrow \neg fly(x) && \text{[Quails usually do not fly.]} \end{aligned}$$

Since r_2 is more specific than r_1 we want $r_2 > r_1$, meaning that r_2 is preferred over r_1 .

CDL has six proof algorithms. The μ algorithm is monotonic and uses only strict rules. CDL restricted to μ is essentially classical propositional logic. The ambiguity blocking algorithm is denoted by β . There are two ambiguity propagating algorithms denoted by ρ and π , ρ being more reliable than π . Algorithm ρ requires the co-algorithm ι , which only considers supporting evidence for a formula and ignores all contrary evidence. Algorithm π requires the co-algorithm ε , which sees if there is unbeaten evidence for a formula. Let λ be either ρ or π and λ' be its required co-algorithm. Then the more λ' can prove the less λ can prove. So by changing λ' we can make λ more reliable and closer to μ , or less reliable and closer to β .

The task of proving a formula is done by a recursive proof function P . The input to P is the proof algorithm to be used, the formula to be proved, and the background. The background is an initially empty storage bin into which is put all the literals that are currently being proved as P recursively calls itself. The purpose of this background is to detect loops. The output of P is one of the following proof-values +1, 0, or -1. The +1 means that the formula is proved in a finite number of steps, 0 means that the proof got into a loop which was detected in a finite number of steps, and -1 means that in a finite number of steps it has been demonstrated that there is no proof of the formula and that this demonstration does not get into a loop.

Some proofs use the finite failure of other proofs. Proofs with a proof-value of -1 are always usable. Moreover CDL determines which proofs with a proof-value of 0 are usable and which are not.

4 The Language of Clausal Defeasible Logic (CDL)

CDL uses a countable propositional language with not (\neg), and (\wedge), and or (\vee). The set of all clauses that are resolution-derivable from a set C of clauses is denoted by $Res(C)$. The complement of a formula and the complement of a set of formulas is now defined.

Definition 1. (*complement, \sim*).

C1) If a is an atom then $\sim a$ is $\neg a$.

C2) If f is a formula then $\sim \neg f$ is f .

C3) If F is a set of formulas then $\sim F = \{\sim f : f \in F\}$.

C4) If F is a finite set of formulas then $\sim \wedge F$ is $\vee \sim F$; and $\sim \vee F$ is $\wedge \sim F$.

Definition 2. (*subclause, proper subclause*). A clause $\vee L$ is a subclause of the clause $\vee M$, denoted $\vee L \leq \vee M$, iff $L \subseteq M$. A clause $\vee L$ is a proper subclause of the clause $\vee M$, denoted $\vee L < \vee M$, iff $L \subset M$.

Definition 3. (*rule*). A *rule*, r , is a triple $(A(r), \text{arrow}(r), c(r))$, where $A(r)$ is a finite set of literals called the set of *antecedents* of r , $\text{arrow}(r) \in \{\rightarrow, \Rightarrow, \rightsquigarrow\}$, and $c(r)$ is a literal called the *consequent* of r .

Strict rules use the *strict arrow*, \rightarrow , and are written $A(r) \rightarrow c(r)$.

Defeasible rules use the *defeasible arrow*, \Rightarrow , and are written $A(r) \Rightarrow c(r)$.

Warning rules use the *warning arrow*, \rightsquigarrow , and are written $A(r) \rightsquigarrow c(r)$.

Definition 4. ($R_s, R_d, R_w, R[l], R[L], Cl(R_s), Ru(C)$). Let R be any set of rules, C be any set of clauses, L be any set of literals, and l be any literal.

$R_s = \{r \in R : r \text{ is strict}\}$.

$R[l] = \{r \in R : l = c(r)\}$.

$R_d = \{r \in R : r \text{ is defeasible}\}$.

$R[L] = \{r \in R : c(r) \in L\}$.

$R_w = \{r \in R : r \text{ is a warning rule}\}$.

$Cl(R_s) = \{\vee[\{c(r)\} \cup \sim A(r)] : r \in R_s\}$.

$Ru(C) = \{\sim(L - \{l\}) \rightarrow l : l \in L \text{ and } \vee L \in C\}$.

A strict rule can be converted to a clause, which is what Cl does. Conversely a clause with n literals can be converted to n strict rules, which is what Ru does.

Definition 5. (*cyclic, acyclic*). A binary relation, $>$, on any set R is *cyclic* iff there exists a sequence, (r_1, r_2, \dots, r_n) where $n \geq 1$, of elements of R such that $r_1 > r_2 > \dots > r_n > r_1$. A relation is *acyclic* iff it is not cyclic.

Definition 6. (*priority relation, $>$, $R[l; s]$*). If R is a set of rules then $>$ is a *priority relation* on R iff $>$ is an acyclic binary relation on $R_d \cup R_w$.

$R[l; s] = \{t \in R[l] : t > s\}$.

We read $t > s$ as t has a higher priority than s , or t is superior to s . Notice that strict rules are never superior to or inferior to any rule; and $>$ does not have to be transitive.

Let C be a set of clauses. We want to remove from C all the clauses which are empty, or tautologies, or are proper superclauses of other clauses. The result is called the reduct of C , $Red(C)$, and is formally defined below.

Definition 7. ($Red(C)$, reduct). Let C be a set of clauses. $Red(C)$ is the *reduct* of C iff it satisfies RC1, RC2, and RC3 below.

RC1) $Red(C) \subseteq \{\vee L \in C : L \neq \{\}\}$ and $\vee L$ is not a tautology.

RC2) If $\{\vee L, \vee M\} \subseteq Red(C)$ then $\vee L$ is not a proper subclause of $\vee M$.

RC3) If $\vee L \in C - Red(C)$ then either .1) $\vee L$ is empty or a tautology;

or .2) a proper subclause of $\vee L$ is in $Red(C)$.

Given a set R of rules, $Pre(R, L)$, the predecessors of L , is the set of all literals that could affect any literal in the set L of literals. $IPre(R, L)$ is the set of immediate predecessors of L .

Definition 8. ($Pre(R, \cdot)$). Suppose R is a set of rules, L is a finite set of literals, l is a literal, and $i \in \mathbb{N}$.

$$\begin{aligned} IPre(R, L) &= \bigcup \{A(r) \cup \sim A(r) : r \in R[L \cup \sim L]\}. \\ IPre^0(R, L) &= L \cup \sim L. & IPre^{i+1}(R, L) &= IPre(R, IPre^i(R, L)). \\ Pre(R, L) &= \bigcup \{IPre^i(R, L) : i \in \mathbb{N}\}. & Pre(R, l) &= Pre(R, \{l\}). \end{aligned}$$

Definition 9. (clausal defeasible theory, cdt). Let R be a set of rules. The ordered pair $(R, >)$ is called a *clausal defeasible theory* (cdt) iff DT1, DT2, and DT3 all hold.

- DT1) $R_s = Ru(Red(Res(Cl(R_s))))$.
DT2) $>$ is a priority relation on $R_d \cup R_w$.
DT3) For all literals, l , $Pre(R, l)$ is finite.

The following notation will be needed later.

Definition 10. ($R_s[L, 2]$, $A^*(R_s[l])$). Let $(R, >)$ be a cdt.

$R_s[L, 2] = \{r \in R_s[L] : A(r) \cap \sim L \neq \{\}\}$. If $\{\} \rightarrow l \in R_s$ or $\{\} \rightarrow \sim l \in R_s$ then $A^*(R_s[l]) = \{A(r) : r \in R_s[l]\}$; else $A^*(R_s[l]) = \{A(r) : r \in R_s[l]\} \cup \{\{l\}\}$.

5 Clausal Defeasible Logic (CDL)

To define the proof algorithms of CDL we defined the proof function P , which is done by using the following nine auxiliary functions: *Plaus* (Plausible), *For*, *Nulld* (Nullified), *Discred* (Discredited), *Dftd* (Defeated), *Disql* (Disqualified), *Evid* (Evidence), *Imp* (Impotent), and *Unwin* (Unwinnable). All these functions depend on the cdt $\Theta = (R, >)$, and have their output in $\{+1, 0, -1\}$.

For non-empty sets max and min have their usual meaning. But we also define $\max\{\} = -1$ and $\min\{\} = +1$.

The proof algorithms are explained after their formal definition. In the following C is a finite set of clauses, B and L are finite sets of literals, and l is a literal. All the proof algorithms are the same for conjunctions of clauses, disjunctions of literals, and literals in B .

Definition 11. (P for conjunctions, disjunctions, $l \in B$).

Suppose $\lambda \in \{\mu, \rho, \pi, \beta, \varepsilon, \iota\}$.

$\lambda \wedge$) $P(\lambda, \wedge C, B) = \min\{P(\lambda, c, B) : c \in C\}$.

$\lambda \vee 1$) If $\forall L$ is a tautology then $P(\lambda, \forall L, B) = +1$;

$\lambda \vee 2$) else $P(\lambda, \forall L, B) = \max\{P(\lambda, l, B) : l \in L\} \cup \{P(\lambda, \wedge(A(r) - \sim L), B) : r \in R_s[L, 2]\}$.

$\lambda 1$) If $l \in B$ then $P(\lambda, l, B) = 0$.

Definition 12. (μ and ι proof algorithms continued).

$\mu 2$) If $l \notin B$ then $P(\mu, l, B) = \max\{P(\mu, \wedge A(r), \{l\} \cup B) : r \in R_s[l]\}$.

$\iota 2$) If $l \notin B$ then $P(\iota, l, B) = \max\{P(\iota, \wedge A(r), \{l\} \cup B) : r \in R_s[l] \cup R_d[l]\}$.

Definition 13. (ρ , π , and β proof algorithms continued).

Suppose $\lambda \in \{\rho, \pi, \beta\}$ and define $\rho' = \iota$, $\pi' = \varepsilon$, and $\beta' = \beta$.

- $\lambda 2$) If $l \notin B$ then $P(\lambda, l, B) = \max(\{P(\lambda, \wedge A(r), \{l\} \cup B) : r \in R_s[l]\} \cup \{Plaus(\lambda, l, B)\})$.
- $\lambda 3$) $Plaus(\lambda, l, B) = \min(\{For(\lambda, l, B)\} \cup \{Nulld(\lambda, l, B, I) : I \in A^*(R_s[\sim l])\})$.
- $\lambda 4$) $For(\lambda, l, B) = \max\{P(\lambda, \wedge A(r), \{l\} \cup B) : r \in R_d[l]\}$.
- $\lambda 5$) $Nulld(\lambda, l, B, I) = \max\{Discred(\lambda, l, B, q) : q \in I\}$.
- $\lambda 6$) $Discred(\lambda, l, B, q) = \min\{Dftd(\lambda, l, B, s) : s \in R[q]\}$.
- $\lambda 7$) $Dftd(\lambda, l, B, s) = \max(\{P(\lambda, \wedge A(t), \{l\} \cup B) : t \in R_d[l; s]\} \cup \{Disql(\lambda, l, B, s)\})$.
- $\lambda 8$) If $P(\lambda', \wedge A(s), \{l\} \cup B) = 0$ and $l \notin Pre(R, A(s))$ then $Disql(\lambda, l, B, s) = +1$;
- $\lambda 9$) else $Disql(\lambda, l, B, s) = -P(\lambda', \wedge A(s), \{l\} \cup B)$.

Definition 14. (ε proof algorithm continued).

- $\varepsilon 2$) If $l \notin B$ then $P(\varepsilon, l, B) = \max(\{P(\varepsilon, \wedge A(r), \{l\} \cup B) : r \in R_s[l]\} \cup \{Evid(\varepsilon, l, B, r) : r \in R_d[l]\})$.
- $\varepsilon 3$) $Evid(\varepsilon, l, B, r) = \min(\{P(\varepsilon, \wedge A(r), \{l\} \cup B)\} \cup \{Imp(\varepsilon, l, B, r, I) : I \in A^*(R_s[\sim l])\})$.
- $\varepsilon 4$) $Imp(\varepsilon, l, B, r, I) = \max\{Unwin(\varepsilon, l, B, r, q) : q \in I\}$.
- $\varepsilon 5$) $Unwin(\varepsilon, l, B, r, q) = \min\{Disql(\varepsilon, l, B, s) : s \in R[q; r]\}$.
- $\varepsilon 6$) If $P(\pi, \wedge A(s), \{l\} \cup B) = 0$ and $l \notin Pre(R, A(s))$ then $Disql(\varepsilon, l, B, s) = +1$;
- $\varepsilon 7$) else $Disql(\varepsilon, l, B, s) = -P(\pi, \wedge A(s), \{l\} \cup B)$.

Definition 15. ($\Theta(\lambda+)$, $\Theta(\lambda-)$, $\Theta(\lambda 0)$). Suppose Θ is a cdt, P is the proof

function of Θ , f is a formula, and $\lambda \in \{\mu, \rho, \pi, \beta, \varepsilon, \iota\}$. We define

$\Theta(\lambda+) = \{f : P(\lambda, f, \{\}) = +1\}$, $\Theta(\lambda 0) = \{f : P(\lambda, f, \{\}) = 0\}$, and

$\Theta(\lambda-) = \{f : P(\lambda, f, \{\}) = -1\}$.

We shall now give some insight into the above proof algorithms. Note that

min and max behave like quantifiers. If S is a subset of $\{+1, 0, -1\}$ then

$\min(S) = +1$ iff $\forall i \in S (i = +1)$; $\min(S) = -1$ iff $\exists i \in S (i = -1)$;

$\max(S) = +1$ iff $\exists i \in S (i = +1)$; and $\max(S) = -1$ iff $\forall i \in S (i = -1)$.

Suppose $\lambda \in \{\mu, \rho, \pi, \beta, \varepsilon, \iota\}$.

A conjunction of clauses, $\wedge C$, is proved by proving each clause in C . So for $P(\lambda, \wedge C, B)$ to be +1 each $P(\lambda, c, B)$ must be +1, where $c \in C$. Hence $\lambda \wedge$.

If a disjunction of literals, $\vee L$, is a tautology then we declare it proved. Hence

$\lambda \vee 1$. If $\vee L$ is not a tautology then $\vee L$ is proved by either proving at least one literal in L , $\max(\{P(\lambda, l, B) : l \in L\})$, or by generalising the following idea.

Suppose $L = \{a, b, c\}$ and r is the strict rule $\{\sim a, d\} \rightarrow c$. Then the clausal form of r is $\vee\{\sim d, a, c\}$. So proving d will show $\vee\{a, c\}$ and hence $\vee L$. Putting these two parts together gives $\lambda \vee 2$.

Proving literals is much more involved. If the literal to be proved, l , is in the background B then we are already in the process of trying to prove l . So we are now in a loop. Hence $P(\lambda, l, B) = 0$ and so $\lambda 1$. So suppose that the literal to be proved, l , is not in the background B .

Since μ uses only strict rules, to prove l we must prove the conjunction of the antecedent of a strict rule whose consequent is l . Hence $\mu 2$.

The ι algorithm only considers evidence that supports l and ignores all evidence against l . So to prove l we must prove the conjunction of the antecedent of any strict or defeasible rule whose consequent is l , $\max\{P(\iota, \wedge A(r), \{l\} \cup B) : r \in R_s[l] \cup R_d[l]\}$. Hence $\iota 2$.

Now consider the ρ , π , and β algorithms. Suppose $\lambda \in \{\rho, \pi, \beta\}$ and recall that $\rho' = \iota$, $\pi' = \varepsilon$, and $\beta' = \beta$. To prove l we must either prove the conjunction of the antecedent of a strict rule whose consequent is l , $\max\{P(\lambda, \wedge A(r), \{l\} \cup B) : r \in R_s[l]\}$, or do something else, $\{Plaus(\lambda, l, B)\}$. Hence $\lambda 2$. Note that when we prove the antecedent, $\wedge A(r)$, l must be added to the background. If we are not using a strict rule then we need evidence for l , $For(\lambda, l, B)$, and we need to nullify all the evidence against l , $\{Nullid(\lambda, l, B, I) : I \in A^*(R_s[\sim l])\}$. Hence $\lambda 3$. Each I in $A^*(R_s[\sim l])$ is a set of literals which are inconsistent with l . This gives us the general conflict property.

The evidence for l is established by proving the conjunction of the antecedent of a defeasible rule whose consequent is l . Hence $\lambda 4$.

The evidence against l is nullified by, for each I in $A^*(R_s[\sim l])$, finding a literal, q , in I such that every rule, s , whose consequent is q is defeated. Hence $\lambda 5$ and $\lambda 6$. A rule, s , is defeated by either disqualifying it, $Disql(\lambda, l, B, s)$, or by using team defeat. That is by finding a defeasible rule t whose consequent is l (a member of the team for l) and which is superior to s , and then proving the conjunction of the antecedent of t , $\{P(\lambda, \wedge A(t), \{l\} \cup B) : t \in R_d[l; s]\}$. Hence $\lambda 7$. A rule, s , is disqualified by either using the λ' algorithm to disprove its antecedent, $\lambda 9$; or by showing that using λ' to prove its antecedent loops, $P(\lambda', \wedge A(s), \{l\} \cup B) = 0$, and that the loop does not involve l , $l \notin Pre(R, A(s))$. Hence $\lambda 8$.

Finally consider the ε proof algorithm. To prove l we must either prove the conjunction of the antecedent of a strict rule whose consequent is l , $\max\{P(\varepsilon, \wedge A(r), \{l\} \cup B) : r \in R_s[l]\}$, or find a defeasible rule, r , whose consequent is l , which is sufficient evidence for l , $\{Evid(\varepsilon, l, B, r) : r \in R_d[l]\}$. Hence $\varepsilon 2$. If we are not using a strict rule then we must prove the conjunction of the antecedent of r and make all the evidence against l impotent, $\{Imp(\varepsilon, l, B, r, I) : I \in A^*(R_s[\sim l])\}$. Hence $\varepsilon 3$. The evidence against l is made impotent by, for each I in $A^*(R_s[\sim l])$, finding a literal, q , in I such that every rule, s , whose consequent is q and which is superior to r is disqualified. Hence $\varepsilon 4$ and $\varepsilon 5$. A rule, s , is disqualified by either using π to disprove its antecedent, $\varepsilon 7$; or by showing that using π to prove its antecedent loops, $P(\pi, \wedge A(s), \{l\} \cup B) = 0$, and that the loop does not involve l , $l \notin Pre(R, A(s))$. Hence $\varepsilon 6$.

6 Example

Consider the following example of a negative loop given in Section 2. $R = \{r_a: \{\} \Rightarrow a, r_b: \{\} \Rightarrow b, r_c: \{\} \Rightarrow c, r_{ab}: \{a\} \Rightarrow \neg b, r_{ba}: \{b\} \Rightarrow \neg a, r_{ac}: \{a\} \Rightarrow \neg c\}$. We show that the ambiguity blocking algorithm β proves c , $P(\beta, c, \{\}) = +1$; but that the ambiguity propagating algorithm π disproves c , $P(\pi, c, \{\}) = -1$. Define the cdt Θ by $\Theta = (R, >)$, where $>$ is empty. For each literal l , $A^*(R_s[l]) = \{\{l\}\}$. Also $Pre(R, a) = \{a, \neg a, b, \neg b\} = Pre(R, b)$.

Calculation C1

- 1) $P(\beta, c, \{\}) = \text{Plaus}(\beta, c, \{\})$, by $\beta 2$
- 2) $= \min\{\text{For}(\beta, c, \{\}), \text{Nulld}(\beta, c, \{\}, \{\neg c\})\}$, by $\beta 3$
- 3) $\text{For}(\beta, c, \{\}) = P(\beta, \wedge\{\}, \{c\})$, by $\beta 4$
- 4) $= \min\{\} = +1$, by $\beta \wedge$.
- 5) $\therefore P(\beta, c, \{\}) = \text{Nulld}(\beta, c, \{\}, \{\neg c\})$, by 4, 3, 2, 1.
- 6) $= \text{Discred}(\beta, c, \{\}, \neg c)$, by $\beta 5$
- 7) $= \text{Dftd}(\beta, c, \{\}, r_{ac})$, by $\beta 6$
- 8) $= \text{Disql}(\beta, c, \{\}, r_{ac})$, by $\beta 7$
- 9) $= +1$, by C1.1(9), $c \notin \text{Pre}(R, a)$, $\beta 8$

Calculation C1.1

- 1) $P(\beta, a, \{c\}) = \text{Plaus}(\beta, a, \{c\})$, by $\beta 2$
- 2) $= \min\{\text{For}(\beta, a, \{c\}), \text{Nulld}(\beta, a, \{c\}, \{\neg a\})\}$, by $\beta 3$
- 3) $\text{For}(\beta, a, \{c\}) = P(\beta, \wedge\{\}, \{a, c\})$, by $\beta 4$
- 4) $= \min\{\} = +1$, by $\beta \wedge$.
- 5) $\therefore P(\beta, a, \{c\}) = \text{Nulld}(\beta, a, \{c\}, \{\neg a\})$, by 4, 3, 2, 1.
- 6) $= \text{Discred}(\beta, a, \{c\}, \neg a)$, by $\beta 5$
- 7) $= \text{Dftd}(\beta, a, \{c\}, r_{ba})$, by $\beta 6$
- 8) $= \text{Disql}(\beta, a, \{c\}, r_{ba})$, by $\beta 7$
- 9) $= -0 = 0$, by C1.1.1(9), $a \in \text{Pre}(R, b)$, $\beta 9$

Calculation C1.1.1

- 1) $P(\beta, b, \{a, c\}) = \text{Plaus}(\beta, b, \{a, c\})$, by $\beta 2$
- 2) $= \min\{\text{For}(\beta, b, \{a, c\}), \text{Nulld}(\beta, b, \{a, c\}, \{\neg b\})\}$, by $\beta 3$
- 3) $\text{For}(\beta, b, \{a, c\}) = P(\beta, \wedge\{\}, \{b, a, c\})$, by $\beta 4$
- 4) $= \min\{\} = +1$, by $\beta \wedge$.
- 5) $\therefore P(\beta, b, \{a, c\}) = \text{Nulld}(\beta, b, \{a, c\}, \{\neg b\})$, by 4, 3, 2, 1.
- 6) $= \text{Discred}(\beta, b, \{a, c\}, \neg b)$, by $\beta 5$
- 7) $= \text{Dftd}(\beta, b, \{a, c\}, r_{ab})$, by $\beta 6$
- 8) $= \text{Disql}(\beta, b, \{a, c\}, r_{ab})$, by $\beta 7$
- 9) $= -0 = 0$, by C1.1.1.1(1), $b \in \text{Pre}(R, a)$, $\beta 9$

Calculation C1.1.1.1

- 1) $P(\beta, a, \{b, a, c\}) = 0$, by $\beta 1$.

Calculation C2

- 1) $P(\pi, c, \{\}) = \text{Plaus}(\pi, c, \{\})$, by $\pi 2$
- 2) $= \min\{\text{For}(\pi, c, \{\}), \text{Nulld}(\pi, c, \{\}, \{\neg c\})\}$, by $\pi 3$
- 3) $\text{For}(\pi, c, \{\}) = P(\pi, \wedge\{\}, \{c\})$, by $\pi 4$
- 4) $= \min\{\} = +1$, by $\pi \wedge$.
- 5) $\therefore P(\pi, c, \{\}) = \text{Nulld}(\pi, c, \{\}, \{\neg c\})$, by 4, 3, 2, 1.
- 6) $= \text{Discred}(\pi, c, \{\}, \neg c)$, by $\pi 5$
- 7) $= \text{Dftd}(\pi, c, \{\}, r_{ac})$, by $\pi 6$
- 8) $= \text{Disql}(\pi, c, \{\}, r_{ac})$, by $\pi 7$
- 9) $= - + 1 = -1$, by C2.1(6), $\pi 9$

Calculation C2.1

- 1) $P(\varepsilon, a, \{c\}) = \text{Evid}(\varepsilon, a, \{c\}, r_a)$, by $\varepsilon 2$
- 2) $= \min\{P(\varepsilon, \wedge\{\}, \{a, c\}), \text{Imp}(\varepsilon, a, \{c\}, r_a, \{\neg a\})\}$, by $\varepsilon 3$

- 3) $P(\varepsilon, \wedge\{c\}, \{a, c\}) = \min\{c\} = +1$, by $\varepsilon\wedge$.
- 4) $\therefore P(\varepsilon, a, \{c\}) = \text{Imp}(\varepsilon, a, \{c\}, r_a, \{\neg a\})$, by 3, 2, 1.
- 5) $\quad = \text{Unwin}(\varepsilon, a, \{c\}, r_a, \neg a)$, by $\varepsilon 4$
- 6) $\quad = \min\{c\} = +1$, by $\varepsilon 5$

This example shows how π and β differ, and also how the failure-by-looping mechanism works.

7 Results

Our first result shows that the proof function P and its auxiliary functions really are functions, and that termination and weak decisiveness also hold.

Theorem 1. Let $\Theta = (R, >)$ be a cdt. If $fn \in \{P, \text{Plaus}, \text{For}, \text{Nulld}, \text{Discred}, \text{Dftd}, \text{Disql}, \text{Evid}, \text{Imp}, \text{Unwin}\}$ then fn is a function with co-domain $\{+1, 0, -1\}$.

The proof function P can be defined without using the auxiliary functions. This re-phrasing of P has a similar structure to the definitions used in DL93, and enables P to be written as many complicated conventional inference rules.

The importance of having a linear proof hierarchy was explained in Section 2. The next theorem says that CDL has this property, and also the reverse for disproof.

Theorem 2. Suppose Θ is a cdt.

- (1) $P(\mu, f, B) \leq P(\rho, f, B) \leq P(\pi, f, B) \leq P(\beta, f, B) \leq P(\varepsilon, f, B) \leq P(\iota, f, B)$.
- (2) $\Theta(\mu+) \subseteq \Theta(\rho+) \subseteq \Theta(\pi+) \subseteq \Theta(\beta+) \subseteq \Theta(\varepsilon+) \subseteq \Theta(\iota+)$.
- (3) $\Theta(\iota-) \subseteq \Theta(\varepsilon-) \subseteq \Theta(\beta-) \subseteq \Theta(\pi-) \subseteq \Theta(\rho-) \subseteq \Theta(\mu-)$.

The following theorem shows that if the priority relation is empty then $\varepsilon = \iota$ and $\pi = \rho$.

Theorem 3. Suppose $(R, >)$ is a cdt where $>$ is empty.

- (1) $P(\varepsilon, f, B) = P(\iota, f, B)$.
- (2) $P(\pi, f, B) = P(\rho, f, B)$.

Except for ε , if the antecedent of a strict rule is proved then its consequent can also be proved; and so modus ponens or detachment holds for strict rules.

Theorem 4. Suppose $\Theta = (R, >)$ is a cdt, $\lambda \in \{\mu, \rho, \pi, \beta, \iota\}$, and $A \rightarrow l$ is in R_s . If $\wedge A \in \Theta(\lambda+)$ then $l \in \Theta(\lambda+)$.

The following example shows that modus ponens need not hold for ε and strict rules. Define $(R, >)$ as follows. $R = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7\}$ where r_1 is $\{a, b\} \rightarrow l$, r_2 is $\{a, \neg l\} \rightarrow \neg b$, r_3 is $\{b, \neg l\} \rightarrow \neg a$, r_4 is $\{\} \Rightarrow b$, r_5 is $\{\} \Rightarrow \neg b$, r_6 is $\{\} \Rightarrow a$, r_7 is $\{l\} \Rightarrow \neg a$; and $>$ is defined by $r_7 > r_6$. Then $P(\varepsilon, a, \{\}) = +1$, and $P(\varepsilon, b, \{\}) = +1$, but $P(\varepsilon, l, \{\}) = 0$. So although the antecedent of r_1 is proved its consequent is not. Hence modus ponens fails for ε and r_1 .

Definition 16. (Consistent). A set C of clauses is *consistent* iff $\vee\{c\} \notin \text{Res}(C)$. C is *inconsistent* iff C is not consistent. $\Theta(\lambda+)$ is *consistent* iff the set of clauses in $\Theta(\lambda+)$ is consistent. $\Theta(\lambda+)$ is *inconsistent* iff $\Theta(\lambda+)$ is not consistent.

Our next result says that our proof algorithms are at least as powerful as resolution; in the sense that if two clauses can be proved then their resolvent can also be proved.

Theorem 5. Suppose $\Theta = (R, >)$ is a cdt, and $\lambda \in \{\mu, \rho, \pi, \beta\}$. If $Cl(R_s)$ is consistent then $\Theta(\lambda+)$ is closed under resolution.

Our final result shows that the proof algorithms do not create inconsistencies, and so are trustworthy.

Theorem 6. If $\Theta = (R, >)$ is a cdt, and $\lambda \in \{\mu, \rho, \pi, \beta\}$ then $\Theta(\lambda+) \cup Cl(R_s)$ is consistent iff $Cl(R_s)$ is consistent.

8 Summary

In Section 1 we argued that, among non-monotonic logics, the family of defeasible logics is important for knowledge representation and reasoning. Defeasible logics are powerful enough for a diverse range of practical applications, and yet their language has a unique combination of expressiveness, simplicity, and naturalness.

However all previous defeasible logics have various faults and so give unintuitive answers for some reasoning puzzles. So a defeasible logic that does not suffer from these faults is needed. Clausal defeasible logic (CDL) is such a logic, and yet it inherits all the desirable properties of the family of defeasible logics. This paper defines and explains CDL, gives an example of how to calculate with both the ambiguity blocking and ambiguity propagating algorithms, and shows that CDL is well-behaved and that its deduction algorithms are trustworthy.

References

1. Antoniou, G. 2002. Nonmonotonic rule system on top of ontology layer. ISWC 2002. LNCS 2432: 394-398. Springer.
2. Antoniou, G.; Billington, D.; Governatori, G.; and Maher, M. J. 2000. A Flexible Framework for Defeasible Logics. Proc AAAI 2000: 405-410.
3. Antoniou, G.; Billington, D.; and Maher, M. J. 1999. On the analysis of regulations using defeasible rules. Proc 32nd Hawaii Intl Conf on Syst Sci (HICSS). IEEE Press.
4. Bassiliades, N.; Antoniou, G.; and Vlahavas, I. 2004. DR-DEVICE: A defeasible logic system for the Semantic Web. Semantic Web Reasoning. LNCS 3208: 134-148. Springer.
5. Billington, D. 1993. Defeasible Logic is Stable. J Logic Computation 3(4): 379-400.
6. Billington, D. 2004. A Plausible Logic which Detects Loops. Proc 10th International Workshop on Nonmonotonic Reasoning: 65-71. ISBN 92-990021-0-X
7. Billington, D. 2005. The Proof Algorithms of Plausible Logic Form a Hierarchy. Proc 18th Australian Joint Conf on AI. LNAI 3809: 796-799. Springer.
8. Billington, D.; and Rock, A. 2001. Propositional Plausible Logic: Introduction and Implementation. Studia Logica 67(2): 243-269.

9. Billington, D.; Estivill-Castro, V.; Hexel, R.; and Rock, A. 2005. Non-monotonic Reasoning for Localisation in RoboCup. Proc 2005 Australasian Conference on Robotics and Automation. At <http://www.cse.unsw.edu.au/~acra2005/proceedings/papers/billington.pdf>
10. Billington, D.; Estivill-Castro, V.; Hexel, R.; and Rock, A. 2007. Using Temporal Consistency to Improve Robot Localisation. RoboCup. LNAI 4434: 232-244. Springer.
11. Darcy, P.; Stantic, B.; and Derakhshan, R. 2007. Correcting Stored RFID Data with Non-Monotonic Reasoning. Int J of Principles and Apps of Info Sci and Tech (PAIST) 1(1): 65-77.
12. Governatori, G. 2005. Representing business contracts in RuleML. International Journal of Cooperative Information Systems 14(2-3): 181-216.
13. Governatori, G.; Dumas, M.; ter Hofstede, A. H.; and Oaks, P. 2001. A formal approach to protocols and strategies for (legal) negotiation. Proc 8th International Conference on Artificial Intelligence and Law (ICAIL01), 168-177. ACM Press.
14. Governatori, G.; Hulstijn, J.; Riveret, R.; and Rotolo, A. 2007. Characterising Deadlines in Temporal Modal Defeasible Logic. AJCAI. LNAI 4830: 486-496. Springer.
15. Governatori, G.; and Padmanabhan, V. 2003. A defeasible logic of policy-based intention. In AI 2003: Advances in AI. LNAI 2903: 414-426. Springer.
16. Governatori, G.; and Rotolo, A. 2004. Defeasible logic: Agency, intention and obligation. In Deontic Logic in Computer Science. LNAI 3065: 114-128. Springer.
17. Governatori, G.; Rotolo, A.; and Sadiq, S. 2004. A model of dynamic resource allocation in workflow systems. Database Technology 2004. Research & Practice of IT, 27: 197-206.
18. Governatori, G.; Rotolo, A.; and Sartor, G. 2005. Temporalised normative positions in defeasible logic. 10th Intl Conf on AI and Law (ICAIL05), 25-34. ACM Press.
19. Johnston, B.; and Governatori, G. 2003. An algorithm for the induction of defeasible logic theories from databases. Database Technology 2003. Research & Practice of IT, 17: 75-83.
20. Maher, M.J.; Rock, A.; Antoniou, G.; Billington, D.; and Miller, T. 2001. Efficient Defeasible Reasoning Systems. Intl J of Artificial Intelligence Tools 10(4): 483-501.
21. Maier, F.; and Nute, D. 2006. Ambiguity Propagating Defeasible Logic and the Well-Founded Semantics. JELIA2006, LNAI 4160: 306-318. Springer.
22. Nute, D. 1987. Defeasible reasoning. 20th Hawaii Intl Conf on Syst Sci, 470-477.
23. Nute, D. 2003. Defeasible Logic. Proc 14th International Conference on Applications of Prolog (INAP2001) LNAI 2543: 151-169. Springer.
24. Reiter, R. 1980. A Logic for Default Reasoning. Artificial Intelligence 13: 81-132.
25. Thakur, S.; Governatori, G.; Padmanabhan, V.; and Lundstrom, J. E. 2007. Dialogue Games in Defeasible Logic. AJCAI. LNAI 4830: 497-506. Springer.