A Plausible Logic which Detects Loops

David Billington

School of Computing and Information Technology
Nathan Campus, Griffith University
Brisbane, Queensland, 4111, Australia
e-mail: D.Billington@griffith.edu.au

Abstract

Unlike most non-monotonic logics Plausible Logic was designed from the very beginning with computer implementation in mind. But one aspect of implementation was neglected, namely loop detection. When using Plausible Logic choices must be made. Typically all choices are explored and then the best one is chosen. If one choice loops then this needs to be detected so that the other choices can be explored. The version of Plausible Logic defined in this paper is called Loop-detecting Plausible Logic (LPL), and it explicitly builds the detection of loops into the logic. This paper presents LPL, considers looping in a slightly more general context, shows that LPL is well-behaved and that looping is unavoidable.

1. Introduction

In the late 1980s Nute (Nute 1987) introduced a non-monotonic reasoning formalism called Defeasible Logic. In the late 1990s Billington (Billington 1998) introduced Plausible Logic which was based on Defeasible Logic. Both formalisms are propositional and deal with defeasible information, as well as factual information. Both logics have a simple rule based syntax; distinguish between formulas proved from just the facts and those proved using defeasible information; and have a constructive deduction procedure which is easy to implement (Billington and Rock 2001). The main difference between the two logics is that Plausible Logic can represent and prove clauses, whereas Defeasible Logic cannot.

Unlike most non-monotonic logics, Plausible Logic was designed from the very beginning with computer implementation in mind. But one aspect of implementation was neglected, namely loop detection. When using Plausible Logic choices must be made. Typically all choices are explored and then the best one is chosen. If one choice loops then this needs to be detected so that the other choices can be explored. The version of Plausible Logic defined in this paper is called Loop-detecting Plausible Logic (LPL), and it explicitly builds the detection of loops into the logic. It would be possible to remove the loop detection from LPL. The resulting logic would have the same set of conclusions as LPL.

The purpose of this paper is to present LPL, and to consider looping in a slightly more general context. An overview of Plausible Logic is given in section 2. Section 3 defines and explains the language of LPL, and section 4 defines and explains LPL. Section 5 presents an example. Section 6 states results that show that LPL is well-behaved. Section 7 investigates looping in a more general context and gives an example for which looping is unavoidable. Section 8 is the conclusion. Sections 3 and 4 formally define LPL and so are more difficult to read. A top level understanding of this paper may be gained by omitting sections 3 to 5 and the proofs in section 6 and referring back to them only when required.

2. An Overview of Plausible Logic

Before the formal definition of Plausible Logic in sections 3 and 4, it may be helpful to give an informal overview of Plausible Logic. The factual and defeasible information with which Plausible Logic reasons is represented by strict rules, plausible rules, defeater rules, and a priority relation on the rules. A rule which contains free variables is treated as a rule schema, and so regarded as an abbreviation for the set of ground instances of the rule. Thus Plausible Logic is essentially propositional. All rules have the form set-of-literals arrow literal.

Strict rules, for example $A \rightarrow l$, represent the aspects of a situation which are certain. If all the literals in A are proved then l can be deduced, no matter what the evidence against l is. So strict rules behave like material implication. An atomic fact is represented by a strict rule with an empty antecedent. For example, "Emma is an emu." is represented by $\{\} \rightarrow emu(emma)$.

More generally a clausal fact with n literals is represented by n strict rules. For instance, $\vee\{a, b, c\}$ is represented by the following three strict rules: $\{\sim b, \sim c\} \rightarrow a$, $\{\sim a, \sim c\} \rightarrow b$, $\{\sim a, \sim b\} \rightarrow c$. All factual information must first be transformed into a set of clauses, and then each clause is converted into a set of strict rules. For example, "Emus are birds." is thought of as the strict rule (schema) $emu(x) \rightarrow bird(x)$, which is transformed into the clause $\vee\{\sim emu(x), bird(x)\}$ which is converted into the two strict rules $emu(x) \rightarrow bird(x)$ and its contrapositive $\sim bird(x) \rightarrow \sim emu(x)$. (If the antecedent of a rule is a singleton set then we often omit the set braces.)

Plausible rules, for example $A \Rightarrow l$, represent some of the aspects of a situation which are plausible. If all the literals in A are proved then l can be deduced provided that all the evidence against l has been nullified. So we take $A \Rightarrow l$ to mean that, in the absence of evidence against l, A is sufficient evidence for concluding l. For example, "Birds usually fly." is represented by $bird(x) \Rightarrow fly(x)$. The idea is that if we know that something is a bird, then we may conclude that it flies, unless there is other evidence suggesting that it may not fly. Plausible rules are thus like normal defaults in Default Logic.

A defeater rule, for example $A \rightharpoonup \sim l$, is evidence against l. In the absence of other rules, $A \rightharpoonup \sim l$ means that if A is not disproved then it is too risky to conclude l. Defeater rules are used to prevent conclusions which would be too risky. For example, "Sick birds might not fly." is represented by $\{sick(x), bird(x)\} \rightharpoonup \sim fly(x)$. The idea is that a bird being sick is not sufficient evidence to conclude that it does not fly; it is only evidence against the conclusion that it usually flies. Another use for defeater rules is to cut chains of plausible rules which are too long. For instance, given $a \Rightarrow b$ and $b \Rightarrow c$ it may be too risky to conclude that c holds given that a holds. In which case we could add the defeater rule $a \rightharpoonup \sim c$. The point is that adding $a \Rightarrow \sim c$ instead of $a \rightharpoonup \sim c$ would be wrong, because accepting a is not a reason for accepting $\sim c$, indeed it is a weak reason for accepting c.

The priority relation, >, on the set of rules allows the representation of preferences among rules. The priority relation must be acyclic. Moreover only plausible rules can occur on the left of >; and strict rules must not occur on either side of >. For example consider the following situation.

	$quail(Quin)$	[Quin is a quail.]
	$bird(Quin)$	[Quin is a bird.]
$R1$:	$bird(x) \Rightarrow fly(x)$	[Birds usually fly.]
$R2$:	$quail(x) \Rightarrow \sim fly(x)$	[Quails usually do not fly.]

We want to conclude that usually Quin does not fly. But this can only be done if we prefer $R2$ to $R1$, that is we define $R2 > R1$.

Most non-monmotonic logics do not distinguish between formulas proved using only factual information and those proved using defeasible information. Plausible Logic does. Indeed Plausible Logic distinguishes between formulas proved with each of five different proof algorithms. (Alternatively this could be viewed as five different logics.) It does this by attaching the name of the algorithm to the formula being proved. The five algorithms are denoted by μ, α, π, β, and δ. So instead of proving $\sim fly(Quin)$ we actually prove $\lambda \sim fly(Quin)$, where $\lambda \in \{\mu, \alpha, \pi, \beta, \delta\}$. In most cases these algorithms form the following hierarchy. Everything proved by the μ algorithm is provable by the α algorithm, everything proved by the α algorithm is provable by the π algorithm, everything proved by the π algorithm is provable by the β algorithm, and everything proved by the β algorithm is provable by the δ algorithm. The stronger or more restricted algorithms give conclusions which are more reliable. The μ algorithm only uses factual information. Plausible Logic restricted to just the μ algorithm is similar to classical propositional logic. The other four algorithms use all the available information. The π algorithm propagates ambiguity. The β algorithm blocks ambiguity and has been used in every Defeasible and Plausible Logic. The α algorithm is the conjunction (and) of the π and β algorithms. The δ algorithm is the disjunction of the π and β algorithms. Although the exact relationship between the algorithms is still being investigated, such a hierarchy would partially solve the debate about whether propagating or blocking ambiguity is best.

A plausible theory, $T = (R, >)$, consists of a set of rules R and its priority relation >, which may be empty. The task of proving a formula is done by a recursive function P called the proof function of T. The input to P is the proof algorithm to be used, the formula to be proved, and the empty set. The empty set is an initially empty storage bin into which is put all the literals which are currently being proved as P recursively calls itself. The purpose of this bin is to enable the detection of loops. The output of P is either +1, 0, or −1. Essentially we have a three valued logic in which +1 denotes proved, 0 denotes loops, and −1 denotes that there is a demonstration that the formula is not provable and does not generate a loop.

3. The Language of Plausible Logic

The next two sections formally define Plausible Logic and so are more difficult to read. It may be possible to skip this section, and refer back to it only when the definition of some notation is required.

We often abbreviate "if and only if" by "iff". X is a subset of Y is denoted by $X \subseteq Y$; the notation $X \subset Y$ means $X \subseteq Y$ and $X \neq Y$, and denotes that X is a *proper subset* of Y. The empty set is denoted by $\{\}$, and the set of all integers by $\mathbb{Z}$. If m and n are integers then we define $[m..n] = \{i \in \mathbb{Z} : m \le i \le n\}$. Let S be any set. The cardinality of S is denoted by $|S|$. It is sometimes convenient to abbreviate "for all x in S" by "$\forall x \in S$". Also "there exists an x in S such that" is sometimes abbreviated to "$\exists x \in S$".

Our *alphabet* is the union of the following four pairwise disjoint sets of symbols: a non-empty finite set, Atm, of (propositional) atoms; the set $\{\neg, \wedge, \vee, \rightarrow, \Rightarrow, \rightharpoonup\}$ of connectives; the set $\{\mu, \alpha, \pi, \beta, \delta\}$ of *tags* denoting various proof algorithms; and the set of punctuation marks consisting of the comma and both braces. By a *literal* we mean any atom, a, or its negation, $\neg a$. A *clause*, $\vee L$, is the disjunction of a finite set, L, of literals. $\vee\{\}$ is the *empty clause* or *falsum* and is thought of as always being false. If l is a literal then we regard $\vee\{l\}$ as another notation for l and so each literal is a clause. A clause $\vee L$ is a tautology iff both an atom and its negation are in L. A *contingent* clause is a clause which is not empty and not a tautology. A *dual-clause*, $\wedge L$, is the conjunction of a finite set, L, of literals. $\wedge\{\}$ is the *empty dual-clause* or *verum* and is thought of as always being true. If l is a literal then we regard $\wedge\{l\}$ as another notation for l and so each literal is a dual-clause. Thus $\wedge\{l\} = l = \vee\{l\}$. Neither the verum nor the falsum are literals. The verum is not a clause, and the falsum is not a dual-clause. A *cnf-formula*, $\wedge C$, is the conjunction of a finite set, C, of clauses. A *dnf-formula*, $\vee D$, is the disjunction of a finite set, D, of dual-clauses. If c is a clause then we regard $\wedge\{c\}$ as another notation for c. If d is a dual-clause then we regard $\vee\{d\}$ as another notation for d. Thus both clauses and dual-clauses are both cnf-formulas and dnf-formulas. By a *formula* we mean any cnf-formula or any dnf-formula. The set of all literals is denoted by *Lit*; the set of all clauses is denoted by *Cls*; the set of all dual-clauses is denoted by *DCls*; the set of all cnf-formulas is denoted by *CnfFrm*; the set of all dnf-formulas is denoted by *DnfFrm*; and the set of all formulas is denoted by *Frm*. *Frm* is finite.

We define the *complement*, $\sim f$, of a formula f and the *complement*, $\sim F$, of a set of formulas F as follows. If f is an atom then $\sim f$ is $\neg f$; and $\sim\neg f$ is f. If L is a set of literals then $\sim L = \{\sim l : l \in L\}$. If $\vee L$ is a clause then $\sim\vee L = \wedge\sim L$. If $\wedge L$ is a dual-clause then $\sim\wedge L = \vee\sim L$. So the complement of a clause is a dual-clause, and the complement of a dual-clause is a clause. In particular the falsum and the verum are complements of each other. If E is a set of clauses or a set of dual-clauses then $\sim E = \{\sim e : e \in E\}$. If $\wedge C$ is a cnf-formula then $\sim\wedge C = \vee\sim C$. If $\vee D$ is a dnf-formula then $\sim\vee D = \wedge\sim D$. So the complement of a cnf-formula is a dnf-formula, and the complement of a dnf-formula is a cnf-formula. If F is a set of formulas then $\sim F = \{\sim f : f \in F\}$. Both *Lit* and *Frm* are closed under complementation.

The information with which Plausible Logic reasons is either certain or defeasible. All the information is represented by various kinds of rules and a priority relation on those rules. Define r to be a *rule* iff $r = (A(r), arrow(r), c(r))$ where $A(r)$ is a finite set of literals called the *antecedent* of r, $arrow(r) \in \{\rightarrow, \Rightarrow, \rightharpoonup\}$, and $c(r)$ is a literal called the *consequent* of r. A rule r which contains the *strict arrow*, $\rightarrow$, is called a *strict rule* and is usually written $A(r) \rightarrow c(r)$. A rule r which contains the *plausible arrow*, $\Rightarrow$, is called a *plausible rule* and is usually written $A(r) \Rightarrow c(r)$. A rule r which contains the *defeater arrow*, $\rightharpoonup$, is called a *defeater rule* and is usually written $A(r) \rightharpoonup c(r)$. The antecedent of a rule can be the empty set. The set of all rules is denoted by *Rul*. *Rul* is finite.

Let R be any set of rules. The set of antecedents of R is denote by $A(R)$; that is $A(R) = \{A(r) : r \in R\}$. The set of consequents of R is denote by $c(R)$; that is $c(R) = \{c(r) : r \in R\}$. We denote the set of strict rules in R by R_s, the set of plausible rules in R by R_p, and the set of defeater rules in R by R_d. Also we define $R_{pd} = R_p \cup R_d$ and $R_{sp} = R_s \cup R_p$.

Let l be any literal. If C is any set of clauses define $C[l] = \{\vee L \in C : l \in L\}$. Then $C[l]$ is the set of all clauses in C which contain l. If R is any set of rules and L is any set of literals then define $R[l] = \{r \in R : l = c(r)\}$, and $R[L] = \{r \in R : c(r) \in L\}$. Then $R[l]$ is the set of all rules in R which end with l; and $R[L]$ is the set of all rules in R which have a consequent in L.

Any binary relation, $>$, on any set S is *cyclic* iff there exists a sequence, $(r_1, r_2, ..., r_n)$ where $n \geq 1$, of elements of S such that $r_1 > r_2 > ... > r_n > r_1$. A relation is *acyclic* iff it is not cyclic. If R is a set of rules then $>$ is a *priority relation* on R iff $>$ is an acyclic binary relation on R such that $>$ is a subset of $R_p \times R_{pd}$. We read $r_1 > r_2$ as r_1 *beats* r_2, or r_2 *is beaten by* r_1. Notice that strict rules never beat, and are never beaten by, any rule. Also defeater rules never beat any rule. Let $R[l;s] = \{t \in R[l] : t > s\}$. Then $R[l;s]$ is the set of all rules in R that beat s and have consequent l.

A *plausible description* of a situation is a 4-tuple $PD = (Ax, R_p, R_d, >)$ such that PD1, PD2, PD3, and PD4 all hold.
(PD1) Ax is a set of contingent clauses.
(PD2) R_p is a set of plausible rules.
(PD3) R_d is a set of defeater rules.
(PD4) $>$ is a priority relation on R_{pd}.
The clauses in Ax, called *axioms*, characterise the aspects of the situation that are certain. The set, $SR(Ax)$, of strict rules *generated* from Ax is defined by $SR(Ax) = \{\sim(L - \{l\}) \rightarrow l : l \in L \text{ and } \vee L \in Ax\}$. $(R, >)$ is the *plausible theory generated by* the plausible description $(Ax, R_p, R_d, >)$ iff $R = SR(Ax) \cup R_p \cup R_d$. T is a *plausible theory* iff there is a plausible description which generates T. If $T = (R, >)$ is a plausible theory then define $Ax(T) = \{\vee(\{c(r)\} \cup \sim A(r)) : r \in R_s\}$. Then $SR(Ax(T)) = R_s$.

Let S be a set of clauses. A clause C_n is *resolution-derivable from S* iff there is a finite sequence of clauses $C_1, ..., C_n$ such that for each i in $[1..n]$, either $C_i \in S$ or C_i is the resolvent of two preceding clauses. The sequence $C_1, ..., C_n$ is called a *resolution-derivation* of C_n *from S*. The set of all clauses which are resolution-derivable from S is denoted by $Res(S)$. Define $Rsn(S) = Res(S) - \{\vee\{\}\}$. $Rsn(S)$ is the set of all non-empty clauses in $Res(S)$.

Let S be a set of sets. Define the set of *minimal elements of S*, $Min(S)$, to be the set of minimal elements of the partially ordered set $(S, \subseteq)$. That is, $Min(S) = \{Y \in S : \text{if } X \subset Y \text{ then } X \notin S\}$.

Let $T = (R, >)$ be a plausible theory. Define $Inc(R) = \{\sim L : \vee L \in Rsn(Ax(T)) \cup \{\vee\{k, \sim k\} : k \in c(R)\}\}$. Then every set in $Inc(R)$ is a non-empty set of literals which are *inconsistent with* R. Define $Inc(R,l) = Min(\{I - \{l\} : I \in Inc(R) \text{ and } l \in I\})$. Each member of $Inc(R,l)$ is a minimal set of literals which is *inconsistent with* l. Since Ax is finite and R is finite, $Res(Ax)$, $Rsn(Ax)$, $Inc(R)$, and $Inc(R,l)$ are finite.

A *tagged* formula is a formula preceded by a tag; so all tagged formulas have the form λf where $\lambda \in \{\mu, \alpha, \pi, \beta, \delta\}$, and f is a formula.

4. Plausible Logic

The basic plan for each of the proof algorithms denoted by the various tags is as follows. If C is a set of two or more clauses then to prove $\wedge C$ every member of C must be proved. (See $\wedge$.2 below.) If L is a set of two or more literals then to prove $\vee L$ at least one member of L must be proved. (See $\vee$.2 below.) To prove a literal l two things must be done. First evidence for l must be established. This is done by finding a strict or plausible rule ending in l (see $\mathcal{L}$.1.2) and then proving the conjunction of its antecedent (see $\mathcal{L}$.2 and $\mathcal{L}$.4). The second thing is to nullify all the evidence against l (see $\mathcal{L}$.3.2). In the case of μ there is nothing to be done (see $\mathcal{L}$.3.1). But for the other tags this is done by disabling all the sets, J, of literals which are inconsistent with l (see $\mathcal{L}$.5). A set of literals, J, is disabled by discrediting an element, j, of J (see $\mathcal{L}$.6). Among other things this means that j cannot be proved. A literal, j, is discredited by defeating each rule, s, which ends in j (see $\mathcal{L}$.7). There are two ways to defeat s, either beat it or prove it is inapplicable (see $\mathcal{L}$.8). The first is to find a plausible rule, t, which ends in l and beats s and then prove the conjunction of the antecedent of t (see $\mathcal{L}$.9). The second way to defeat s is to show that s is inapplicable (see $\mathcal{L}$.10). There are four ways to do this as follows. For π we prove the negation of the conjunction of the antecedent of s (see $\mathcal{L}$.10.π). For β we show that all proofs of the conjunction of the antecedent of s fail in a finite number of steps (see $\mathcal{L}$.10.β). For α we show that both the conditions for π and β hold (see $\mathcal{L}$.10.α). For δ we show that at least one of the conditions for π and β holds (see $\mathcal{L}$.10.δ).

So for β, α, and δ showing finite failure is crucial. This means that detecting loops is necessary. To detect loops we introduce a *background* set B of literals whose proof result is currently being calculated. A proof result is either +1 for proved, or 0 for loops, or –1 for finite failure.

Suppose $S \subseteq \{-1, 0, +1\}$. If S is not empty then the minimum element of S is denoted by $\min S$, and the maximum element of S is denoted by $\max S$. We define $\min\{\} = +1$, and $\max\{\} = -1$. However $\{\}$ is the only set whose max is less than its min.

Given a plausible theory $T = (R, >)$ we define the following ten functions P, *Strict*, *Plaus*, *For*, *Nullified*, *Disabled*, *Discredited*, *Defeated*, *Beaten*, and *Inappl* all of which depend on T. P is called the *proof function of* T, the other nine functions merely assist in the definition of P. P takes a tagged cnf-formula λf and a background set B of literals and returns a result in $\{+1, 0, -1\}$. In the following definition, B is a set of literals, l is a literal, and $\lambda \in \{\mu, \alpha, \pi, \beta, \delta\}$.

$\wedge$.1) $P(\lambda\wedge\{\}, B) = +1$.

$\wedge$.2) If C is a set of two or more clauses then
$P(\lambda\wedge C, B) = \min\{P(\lambda c, B) : c \in C\}$.

$\vee$.1) $P(\lambda\vee\{\}, B) = -1$.

$\vee$.2) If L is a set of two or more literals then
$P(\lambda\vee L, B) = \max\{P(\lambda l, B) : l \in L\}$.

$\mathcal{L}$.1.1) If $l \in B$ then $P(\lambda l, B) = 0$.

$\mathcal{L}$.1.2) If $l \notin B$ then
$P(\lambda l, B) = \max\{Strict(\lambda l, B), Plaus(\lambda l, B)\}$.

$\mathcal{L}$.2) $Strict(\lambda l, B) = \max\{P(\lambda\wedge A(r), B \cup \{l\}) : r \in R_s[l]\}$.

$\mathcal{L}$.3.1) $Plaus(\mu l, B) = -1$.

$\mathcal{L}$.3.2) If $\lambda \neq \mu$ then
$Plaus(\lambda l, B) = \min\{For(\lambda l, B), Nullified(\lambda l, B)\}$.

$\mathcal{L}$.4) $For(\lambda l, B) = \max\{P(\lambda\wedge A(r), B \cup \{l\}) : r \in R_p[l]\}$.

$\mathcal{L}$.5) $Nullified(\lambda l, B) =$
$\min\{Disabled(\lambda l, B, J) : J \in Inc(R,l)\}$.

$\mathcal{L}$.6) $Disabled(\lambda l, B, J) =$
$\max\{Discredited(\lambda l, B, j) : j \in J\}$.

$\mathcal{L}$.7) $Discredited(\lambda l, B, j) =$
$\min\{Defeated(\lambda l, B, s) : s \in R[j]\}$.

$\mathcal{L}$.8) $Defeated(\lambda l, B, s) =$
$\max\{Beaten(\lambda l, B, s), Inappl(\lambda l, B, s)\}$.

$\mathcal{L}$.9) $Beaten(\lambda l, B, s) =$
$\max\{P(\lambda\wedge A(t), B \cup \{l\}) : t \in R_p[l;s]\}$.

$\mathcal{L}$.10.α) $Inappl(\alpha l, B, s) =$
$\min\{P(\alpha\sim\wedge A(s), B \cup \{l\}), -P(\alpha\wedge A(s), B \cup \{l\})\}$.

$\mathcal{L}$.10.π) $Inappl(\pi l, B, s) = P(\pi\sim\wedge A(s), B \cup \{l\})$.

$\mathcal{L}$.10.β) $Inappl(\beta l, B, s) = -P(\beta\wedge A(s), B \cup \{l\})$.

$\mathcal{L}$.10.δ) $Inappl(\delta l, B, s) =$
$\max\{P(\delta\sim\wedge A(s), B \cup \{l\}), -P(\delta\wedge A(s), B \cup \{l\})\}$.

The functions *Strict*, *Plaus*, *For*, *Nullified*, *Disabled*, *Discredited*, *Defeated*, *Beaten*, and *Inappl* are undefined if $l \in B$. Similarly the functions *For*, *Nullified*, *Disabled*, *Discredited*, *Defeated*, *Beaten*, and *Inappl* are undefined if $\lambda = \mu$.

The deducibility relation $\vdash$ is defined by $T \vdash +\lambda f$ iff $P(\lambda f, \{\}) = +1$, $T \vdash 0\lambda f$ iff $P(\lambda f, \{\}) = 0$, and $T \vdash -\lambda f$ iff $P(\lambda f, \{\}) = -1$. We define $T(+\lambda) = \{f : T \vdash +\lambda f\}$, $T(0\lambda) = \{f : T \vdash 0\lambda f\}$, and $T(-\lambda) = \{f : T \vdash -\lambda f\}$. A *Plausible Logic* consists of a plausible theory and its proof function.

5. Example

The purpose of this moderately complex example is to illustrate how a reasoning puzzle is translated into a plausible description, how that is translated into a plausible theory, and how the proof function is evaluated.

Suppose

(1) Egbert is an academic computing person. [ac]
(2) Egbert lives in Uniplace. [up]
(3) Academic computing people are computing people. [$ac \rightarrow c$]
(4) Academic computing people are usually not rich. [$ac \Rightarrow \neg r$]
(5) Computing people are usually rich. [$c \Rightarrow r$]
(6) People who live in Uniplace live in Richdale. [$up \rightarrow rd$]
(7) People who live in Uniplace are usually not rich. [$up \Rightarrow \neg r$]
(8) People who live in Richdale are usually rich. [$rd \Rightarrow r$]

Is Egbert rich? [Since $P(p\neg r, \{\}) = +1$, the answer is no.]

A plausible description of this puzzle is $PD = (Ax, R_p, R_d, >)$ where $Ax = \{ac, up, \neg ac \vee c, \neg up \vee rd\}$, $R_p = \{ac \Rightarrow \neg r,\ c \Rightarrow r,\ up \Rightarrow \neg r,\ rd \Rightarrow r\}$, $R_d = \{\}$, and $>$ is defined by $ac \Rightarrow \neg r > c \Rightarrow r$, and $up \Rightarrow \neg r > rd \Rightarrow r$.

Define $R1$ to $R8$ as follows. $R1$: $\{\} \rightarrow ac$, $R2$: $\{\} \rightarrow up$, $R3$: $\{ac\} \rightarrow c$, $R4$: $\{\neg c\} \rightarrow \neg ac$, $R5$: $\{up\} \rightarrow rd$, $R6$: $\{\neg rd\} \rightarrow \neg up$, $R7$: $\{ac\} \Rightarrow \neg r$, $R8$: $\{c\} \Rightarrow r$, $R9$: $\{up\} \Rightarrow \neg r$, $R10$: $\{rd\} \Rightarrow r$.
Then the Plausible Theory corresponding to PD is $T = (R, >)$, where $R = \{R1, ..., R10\}$ and $>$ is defined by $R7 > R8$, and $R9 > R10$.

An evaluation of $P(\pi\neg r, \{\})$ follows. Each step is followed by a justification which either refers to previous steps, or refers to the definition of the function on the left of the = sign. This definition contains variables which are instantiated in the following evaluation. For example $\mathcal{L}.1.2(\lambda{=}\pi, l{=}\neg r, B{=}\{\})$ indicates in the definition prefixed by $\mathcal{L}.1.2$ the λ is replaced by π, the l is replaced by $\neg r$, and the B is replaced by $\{\}$. In the following evaluation since λ is always π we shall not keep mentioning this fact; and we shall abbreviate *Defeated* by *Def*.

1) $P(\pi\neg r, \{\}) = \max\{Strict(\pi\neg r, \{\}), Plaus(\pi\neg r, \{\})\}$,
by $\mathcal{L}.1.2(l{=}\neg r, B{=}\{\})$.
2) $Plaus(\pi\neg r,\{\}) = \min\{For(\pi\neg r,\{\}), Nullified(\pi\neg r,\{\})\}$,
by $\mathcal{L}.3.2(l{=}\neg r, B{=}\{\})$.
3) $For(\pi\neg r,\{\}) = \max\{P(\pi ac,\{\neg r\}), P(\pi up,\{\neg r\})\}$,
by $\mathcal{L}.4(l{=}\neg r, B{=}\{\}, R_p[\neg r]{=}\{R7, R9\})$.
4) $P(\pi ac,\{\neg r\}) =$
$\max\{Strict(\pi ac,\{\neg r\}), Plaus(\pi ac,\{\neg r\})\}$,
by $\mathcal{L}.1.2(l{=}ac, B{=}\{\neg r\})$.
5) $Strict(\pi ac,\{\neg r\}) = P(\pi\wedge\{\},\{ac,\neg r\})$,
by $\mathcal{L}.2(l{=}ac, B{=}\{\neg r\}, R_s[ac]{=}\{R1\})$.
6) $P(\pi\wedge\{\},\{ac,\neg r\}) = +1$, by $\wedge.1(B{=}\{ac,\neg r\})$.
7) $Plaus(\pi\neg r,\{\}) = Nullified(\pi\neg r,\{\})$,
by (6), (5), (4), (3), and (2).
8) $Nullified(\pi\neg r,\{\}) = Disabled(\pi\neg r,\{\},\{r\})$,
by $\mathcal{L}.5(l{=}\neg r, B{=}\{\}, Inc(R,\neg r){=}\{\{r\}\})$.
9) $Disabled(\pi\neg r,\{\},\{r\}) = Discredited(\pi\neg r,\{\},r)$,
by $\mathcal{L}.6(l{=}\neg r, B{=}\{\}, J{=}\{r\})$.
10) $Discredited(\pi\neg r,\{\},r) =$
$\min\{Def(\pi\neg r,\{\},R8), Def(\pi\neg r,\{\},R10)\}$,
by $\mathcal{L}.7(l{=}\neg r, B{=}\{\}, j{=}r, R[r]{=}\{R8, R10\})$.
11) $Def(\pi\neg r,\{\},R8) =$
$\max\{Beaten(\pi\neg r,\{\},R8), Inappl(\pi\neg r,\{\},R8)\}$,
by $\mathcal{L}.8(l{=}\neg r, B{=}\{\}, s{=}R8)$.
12) $Beaten(\pi\neg r,\{\},R8) = P(\pi ac,\{\neg r\})$,
by $\mathcal{L}.9(l{=}\neg r, B{=}\{\}, s{=}R8, R_p[\neg r;R8]{=}\{R7\})$.
13) $P(\pi ac,\{\neg r\}) = +1$, by (6), (5), and (4).
14) $Discredited(\pi\neg r,\{\},r) = Def(\pi\neg r,\{\},R10)$,
by (13), (12), (11), and (10).
15) $Def(\pi\neg r,\{\},R10) =$
$\max\{Beaten(\pi\neg r,\{\},R10), Inappl(\pi\neg r,\{\},R10)\}$,
by $\mathcal{L}.8(l{=}\neg r, B{=}\{\}, s{=}R10)$.
16) $Beaten(\pi\neg r,\{\},R10) = P(\pi up,\{\neg r\})$,
by $\mathcal{L}.9(l{=}\neg r, B{=}\{\}, s{=}R10, R_p[\neg r;R10]{=}\{R9\})$.
17) $P(\pi up,\{\neg r\}) =$
$\max\{Strict(\pi up,\{\neg r\}), Plaus(\pi up,\{\neg r\})\}$,
by $\mathcal{L}.1.2(l{=}up, B{=}\{\neg r\})$.
18) $Strict(\pi up,\{\neg r\}) = P(\pi\wedge\{\},\{up,\neg r\})$,
by $\mathcal{L}.2(l{=}up, B{=}\{\neg r\}, R_s[up]{=}\{R2\})$.
19) $P(\pi\wedge\{\},\{up,\neg r\}) = +1$, by $\wedge.1(B{=}\{up,\neg r\})$.
20) $P(\pi\neg r, \{\}) = +1$, by (19), (18), (17), (16), (15), (14), (9), (8), (7), and (1).

6. Main Results

All the proofs of all the results obtained for LPL are in (Billington 2003a). The first result says that conjunction and disjunction behave as expected. Its proof follows in a straightforward manner from the definitions.

Theorem 1 (Conjunction and Disjunction)
Suppose T is a plausible theory, P is its proof function, and $\lambda \in \{\mu, \alpha, \pi, \beta, \delta\}$. Let C^* be a set of clauses and suppose $C \subseteq C^*$. Let L^* be a set of literals and suppose $L \subseteq L^*$. Let B be a background.
(1) $P(\lambda\wedge C^*, B) \leq P(\lambda\wedge C, B)$.
(2) If $T \vdash +\lambda\wedge C^*$ then $T \vdash +\lambda\wedge C$.
(3) If $T \vdash -\lambda\wedge C$ then $T \vdash -\lambda\wedge C^*$.
(4) $P(\lambda\vee L, B) \leq P(\lambda\vee L^*, B)$.
(5) If $T \vdash +\lambda\vee L$ then $T \vdash +\lambda\vee L^*$.
(6) If $T \vdash -\lambda\vee L^*$ then $T \vdash -\lambda\vee L$.
End

The resolution property (theorem 3) is an interesting property which seems to be necessary for proving relative consistency.

Lemma 2 (Unit Resolution Property)

Suppose T is a plausible theory and $\lambda \in \{\mu, \alpha, \pi, \beta, \delta\}$. Let L be a set of literals and let l be a literal. If $l \in T(+\lambda)$ and $\vee(L\cup\{\sim l\}) \in T(+\lambda)$ then either $\vee L \in T(+\lambda)$ or $\{l, \sim l\} \subseteq T(+\lambda)$.

Proof

Suppose $l \in T(+\lambda)$ and $\vee(L\cup\{\sim l\}) \in T(+\lambda)$. If $|L\cup\{\sim l\}| = 1$ then $L\cup\{\sim l\} = \{\sim l\}$ and so $\sim l \in T(+\lambda)$. Thus $\{l, \sim l\} \subseteq T(+\lambda)$.

So suppose $|L\cup\{\sim l\}| \neq 1$. By $\vee$.2, either $\sim l \in T(+\lambda)$ and so $\{l, \sim l\} \subseteq T(+\lambda)$, or there exists $m \in L-\{\sim l\}$ such that $m \in T(+\lambda)$. If $|L| = 1$ then $L = \{m\}$ and hence $\vee L \in T(+\lambda)$. If $|L| \neq 1$ then by $\vee$.2, $\vee L \in T(+\lambda)$.

EndProof

Theorem 3 (Resolution Property)

Suppose T is a plausible theory and $\lambda \in \{\mu, \alpha, \pi, \beta, \delta\}$. Let L and M be two sets of literals and let l be a literal. If $\vee(L\cup\{l\}) \in T(+\lambda)$ and $\vee(M\cup\{\sim l\}) \in T(+\lambda)$ then either $\vee(L\cup M) \in T(+\lambda)$ or $\{l, \sim l\} \subseteq T(+\lambda)$.

Proof

Suppose $\vee(L\cup\{l\}) \in T(+\lambda)$ and $\vee(M\cup\{\sim l\}) \in T(+\lambda)$. If $|L\cup\{l\}| = 1$ or $|M\cup\{\sim l\}| = 1$ then by lemma 2 and theorem 1(5) the lemma holds. So suppose $|L\cup\{l\}| \neq 1$ and $|M\cup\{\sim l\}| \neq 1$. By $\vee$.2, either $l \in T(+\lambda)$, or there exists $k \in L-\{l\}$ such that $k \in T(+\lambda)$. By $\vee$.2, either $\sim l \in T(+\lambda)$, or there exists $m \in M-\{\sim l\}$ such that $m \in T(+\lambda)$. If $l \in T(+\lambda)$ and $\sim l \in T(+\lambda)$ then $\{l, \sim l\} \subseteq T(+\lambda)$.

So suppose either $l \notin T(+\lambda)$ or $\sim l \notin T(+\lambda)$. Then either $k \in L-\{l\}$ and $k \in T(+\lambda)$, or $m \in M-\{\sim l\}$ and $m \in T(+\lambda)$. In either case there exist $j \in L\cup M$ such that $j \in T(+\lambda)$. If $|L\cup M| = 1$ then $L\cup M = \{j\}$ and hence $\vee(L\cup M) \in T(+\lambda)$. If $|L\cup M| \neq 1$ then by $\vee$.2, $\vee(L\cup M) \in T(+\lambda)$.

EndProof

Let T be a plausible theory and F be a set of cnf-formulas. Define $F^{-}\wedge = \{c : \wedge C \in F \text{ and } c \in C\}$. Then F is *consistent* [respectively, *T-consistent*] iff $\vee\{\} \notin Res(F^{-}\wedge)$ [respectively, $\vee\{\} \notin Res(F^{-}\wedge\cup Ax(T))$]. S is *inconsistent* iff S is not consistent.

If F is T-consistent then F is consistent. But the converse is not always true. For example let $Ax(T) = \{\vee\{a, b\}\}$, and $F = F^{-}\wedge = \{\neg a, \neg b\}$. Then F is consistent but F is not T-consistent.

Theorem 4 (Relative Consistency)

If T is a plausible theory and $\lambda \in \{\mu, \alpha, \pi, \beta, \delta\}$ then $T(+\lambda)$ is T-consistent iff $Ax(T)$ is consistent.

Sketch of Proof

If $T(+\lambda)$ is T-consistent then clearly $Ax(T)$ is consistent.

Conversely, suppose $T(+\lambda)$ is not T-consistent. Then $\vee\{\} \in Res(T(+\lambda)^{-}\wedge\cup Ax)$. Let $T = (R, >)$. If there is a literal l such that $\{l, \sim l\} \subseteq T(+\lambda)^{-}\wedge$, then $\{l, \sim l\} \in Inc(R)$, and so by a lemma Ax is inconsistent. So suppose there is no literal l such that $\{l, \sim l\} \subseteq T(+\lambda)^{-}\wedge$. By theorem 3, $Rsn(T(+\lambda)^{-}\wedge) = T(+\lambda)^{-}\wedge$. If $\vee\{\} \in Res(T(+\lambda)^{-}\wedge)$ then there is a literal k such that $\{k, \sim k\} \subseteq Rsn(T(+\lambda)^{-}\wedge)$, and hence $\{k, \sim k\} \subseteq T(+\lambda)^{-}\wedge$. So suppose $\vee\{\} \notin Res(T(+\lambda)^{-}\wedge)$. Let L be the set of all literals in $T(+\lambda)$. Then $L \subseteq T(+\lambda)^{-}\wedge$, and so by a lemma $\vee\{\} \notin Res(L)$. By $\vee$.2, every clause in $T(+\lambda)^{-}\wedge\cup Ax$ has a subclause in $L\cup Ax$, and so by a lemma every clause in $Res(T(+\lambda)^{-}\wedge\cup Ax)$ has a subclause in $Res(L\cup Ax)$. Hence $\vee\{\} \in Res(L\cup Ax)$. By a lemma there is a finite subset K of L such that $\vee(\sim K) \in Res(Ax)$. If $\sim K = \{\}$ then Ax is inconsistent. So suppose $\sim K$ is not empty. Then $\vee(\sim K) \in Rsn(Ax)$ and so $K \in Inc(R)$. By a lemma, Ax is inconsistent.

EndProof

Relative consistency is important because it shows that the deduction mechanisms do not create inconsistencies, and hence are trustworthy. Unfortunately the more complicated disjunction of the Plausible Logic introduced in (Billington and Rock 2001) has meant that relative consistency could not be proved for that Plausible Logic.

The last two results show that if the axioms of a plausible theory are consistent then the set of proved formulas is closed under resolution.

7. Decisiveness

Consider the following example which we shall call NT for negative triangle. NT consists of the following six plausible rules, the priority relation is empty.

Ra: $\{\} \Rightarrow a$	Rb: $\{\} \Rightarrow b$	Rc: $\{\} \Rightarrow c$
Rab: $a \Rightarrow \neg b$	Rbc: $b \Rightarrow \neg c$	Rca: $c \Rightarrow \neg a$

This example is symmetric in a, b, and c.

All Plausible Logics and all Defeasible Logics have a proof algorithm which has the following property, which we shall call *Prop*. To prove x two conditions must be satisfied. (1) The antecedent $A(r)$ of a strict or plausible rule r whose consequent is x must be proved. (2) If the priority relation is empty then the antecedent $A(s)$ of any rule s whose consequent is $\sim x$ must be proved to be not provable. The β proof algorithm has *Prop*. A proof algorithm is *decisive* iff for each cnf-formula f either f can be proved or f can be proved to be not provable.

Now assume there is a decisive proof algorithm which has *Prop*. In NT if a is provable then b is not provable. By decisiveness b can be proved to be not provable. Therefore c is provable. Thus a is not provable; which is a contradiction. Moreover if a is not provable then by decisiveness a can be proved to be not provable. Therefore b is provable. Therefore c is not provable. By decisiveness c can be proved to be not provable. Thus a is provable; which is a contradiction. Thus decisiveness and *Prop* lead to a contradiction.

Consider the following extension of Plausible Logic by a new proof algorithm denoted by γ. Define γ as follows.

$\mathcal{L}$.10.γ) If $P(\gamma \wedge A(s), B \cup \{l\}) \neq 0$ then
$Inappl(\gamma l, B, s) = -P(\gamma \wedge A(s), B \cup \{l\})$.
If $P(\gamma \wedge A(s), B \cup \{l\}) = 0$ then
$Inappl(\gamma l, B, s) = +1$.

Then γ has *Prop* and is decisive. The attraction of γ is that $P(\gamma f, B)$ is never 0 and so there is no looping. But in the plausible theory defined by NT we have $P(\gamma c, \{\}) = +1$, and $P(\gamma c, \{a\}) = -1$. (Details of the evaluation are in (Billington 2003b).) This contradicts an intuitively appealing lemma of (Billington 2003a) which is used to prove relative consistency. Thus we reject γ, and conclude that *Prop* means that looping is unavoidable.

8. Conclusion

A Loop-detecting Plausible Logic (LPL) has been defined and shown to be well-behaved. A property that is fundamental to the proof algorithms of both Defeasible and Plausible Logic has been shown to make looping unavoidable in some cases.

Loop-detecting Plausible Logic cannot always prove its axioms or tautologies. A slight generalisation of LPL to allow tautologies and LPL's axioms to be proved is being developed. The relationships between the five proof algorithms is being investigated.

Acknowledgements

The initial development of loop checking was joint work with Dr. Andrew Rock.

This paper and I benefitted from the many comments made by Professor Donald Nute while visiting him at the Artificial Intelligence Center of the University of Georgia in February 2004. I would also like to thank the AI Center for their hospitality and working conditions.

References

Billington, D. 1998. Defeasible deduction with arbitrary propositions. In Poster Proceedings of the 11th Australian Joint Conference on Artificial Intelligence, 3-14. Griffith University.

Billington, D. 2003a. Constructive Plausible Logic Version 2.1 Repository. Available from the author.

Billington, D. 2003b. Decisiveness. Available from the author.

Billington, D., and Rock, A. 2001. Propositional Plausible Logic: Introduction and Implementation. *Studia Logica* 67: 243-269.

Nute, D. 1987. Defeasible reasoning. In Proceedings of the 20th Hawaii International Conference on System Science, 470-477. University of Hawaii.