An Implementation of Propositional Plausible Logic

Andrew Rock and David Billington
School of Computing and Information Technology
Griffith University
Nathan, Queensland, 4111, Australia
{A.Rock, D.Billington}@cit.gu.edu.au
Structure:

- Introduction: Plausible Logic is a non-monotonic logic.
- Overview of Plausible Logic
- Implementing Plausible Logic
- Performance Results
- Conclusions and future directions
Non-monotonic reasoning:

• Non-monotonic reasoning is a branch of artificial intelligence which is concerned with the problem of deducing conclusions from incomplete or uncertain information.

• “Non-monotonic” in that conclusions can be revised in the light of new information.

• Most of these logics were not designed with implementation in mind, with poor complexity results. [Gotlob:92]

• More implementations are needed to assess their utility in practical situations.
Defeasible Logic:

- Nute, 1988
- Designed to be implemented
- Has been implemented: d-Prolog.
- The expressivity of Defeasible Logic is limited by its inability to represent or prove disjunctions. (Recent work on the modeling of regulations [Antoniou:99] has shown that the ability to accommodate disjunctions is important.)
Plausible Logic:

- David Billington
- Extends Defeasible Logic, by accommodating disjunction, and hence arbitrary propositions.
- Generalizes the handling of competing rules.
- Allows a more natural representation of some problems, which is important for building confidence that the representation is faithful.
- Increased expressivity, increases the complexity, but for modeling actual regulations the use of disjunctions does not destroy the polynomial nature of the model.
Overview of Plausible Logic

A reasoning situation is defined by a plausible description, made up of:

- a set of indiputable facts, each represented by a formula (examples: furry, \sim furr, furry \lor feathered, furry \land pet, poodle \rightarrow dog);
- a set of plausible rules (example: \{large, common, furry, pet\} \Rightarrow dog) which might have a few exceptions; and
- a set of defeater rules (example: hasMane \sim \sim dog) which can prevent a conclusion without supporting its negation.

In a rule \( r \) the set of formulas on the left of and arrow is the antecedent \( A(r) \) and the formula on the right is the consequent \( c(r) \).
Descriptions enable a very natural representation of reasoning situations, but are difficult to compute with, so they are transformed into Plausible theories.

Plausible theories consist of:

- one set of rules \( R \) with only literals \( (q \) or \( \sim q \)) as consequents (Facts are transformed into strict rules \( \rightarrow \)); and

- a priority relation \( > \) from all rules \( R \) to the plausible and defeater rules \( R_{pd} \). \( > \) must not be cyclic.

From a theory we try to prove formulas at different levels of certainty, using inference conditions defined as follows.
We shall only be concerned with deducing or proving formulas in conjunctive normal (cnf) form. Formulas not in cnf can be transformed into equivalent cnf-formulas prior to their proofs.

A tag \( d \) is any element of \( \{+\Delta, -\Delta, +\delta, -\delta, +\partial, -\partial, +\int, -\int\} \). The tag indicates the desired level of proof.

A tagged cnf-formula is a tag followed by a cnf-formula.

A formal proof or derivation, \( P \), is a finite sequence, \( P = (P(1), \ldots, P(n)) \), of ordered pairs \( (T, \pm df) \) such that \( T \) is a plausible theory. \( P[1..i] \) are the first \( i \) lines of a derivation.
A conjunction is proved only if all of its conjuncts are already proved. Hence the positive conjunctive inference condition $+\land$.

$+\land$ If $P(i + 1) = (T, +d \land F)$ then $\forall f \in F, (T, +df) \in P[1..i]$.

The inference conditions come in pairs, one positive and one negative. The negative one is just the “strong” or “constructive” negative of the positive condition. That is, if $+I$ is a positive inference condition then $-I$ is the negation of $+I$ but with $(\_,-d\_)$ $\notin$ $P[1..i]$ replaced by $(\_,-d\_) \in P[1..i]$, and $(\_,-d\_) \notin P[1..i]$ replaced by $(\_,+d\_) \in P[1..i]$.

A proof of $(T,-df)$ proves that $(T,+df)$ can not be proved.
In considering $\bigvee F$ we can regard $F$ as a set of literals rather than clauses. A disjunction is proved if a non-empty proper subset of its disjuncts is already proved. Also if each literal in $F$ could be proved in the theory formed by adding the complement of all the other literals in $F$ to $T$, then we could add $(T,+d\bigvee F)$ to any proof in $T$.

$+\forall$) If $P(i+1) = (T,+d\bigvee F)$ and $F$ is a set of literals then 

$$\exists F' \in \mathcal{P}_{\geq 1}(F), \forall f \in F',

((R \cup \{\{} \rightarrow \sim q : q \in (F' - \{f\}))}, >), +df) \in P[1..i].$$

where $T = (R, >)$ and $\mathcal{P}_{\geq 1}(F) = \{G : G \subseteq F \text{ and } |G| \geq 1\}$, the set of non-empty subsets of $F$. 
Cnf-formulas can be proved at three different levels of certainty or confidence. In decreasing certainty they are: the definite level $\Delta$, the defeasible level $\partial$ (or $\delta$), and the supported level $\int$.

The definite level is like classical monotonic proof in that modus ponens is used and so more information cannot defeat a previous proof.

If a formula is proved definitely then one should behave as if it is true.

Having dealt with conjunctions and disjunctions we now only need to consider single literals.

$+(\Delta)$ If $P(i + 1) = (T, +(\Delta q))$ then
$$\exists r \in R_s[q] \forall a \in A(r), (T, +(\Delta a)) \in P[1..i].$$

where $R_s[q]$ are the strict rules in $R$ with consequent $q$. 
Proof at the defeasible level is non-monotonic, that is more information may defeat a previous proof. However one should still behave as if a defeasibly proved formula is true, even though this may be wrong.

Before we consider the defeasible level of proof we need to define what we mean by evidence against a literal.

Rules which end with $\sim q$ are direct evidence against $q$, and compete with rules that end with $q$.

However there can be indirect evidence against $q$. Consider the simple strict rule $\{a, q\} \rightarrow c$ and its set $\{a, q, \sim c\}$ of inconsistent literals. A pair of rules, one ending with $a$ and the other ending with $\sim c$, are indirect evidence against $q$, and compete with rules ending with $q$. 
A set of literals is \textit{inconsistent with} $R$ iff it is a member of $I(R)$, where $I(R) = \{A(r) \cup \{\sim c(r)\} : r \text{ is a simple strict rule in } R\} \cup \{\{q, \sim q\} : q \text{ is a literal}\}$.

If $C$ is a set of rules then we denote by $c(C) = \{c(r) : r \in C\}$ the set of consequents of the rules in $C$.

Define $C(R) = \{C \subseteq R : c(C) \in I(R) \text{ and } |c(C)| = |C|\}$. Each member of $C(R)$ is a set of \textit{competing rules}. That is, a set of competing rules is a minimal set of rules whose consequents form an inconsistent set of literals.

Define $C(R, q) = \{S - \{r\} : S \in C(R) \text{ and } r \in S[q]\}$. Each member of $C(R, q)$ is a set of rules which together is evidence against or contrary to $q$. 
The $+\partial$ defeasible inference condition:

$+\partial$) If $P(i + 1) = (T, +\partial q)$ then either

1. $\exists r \in R_s[q] \forall a \in A(r), (T, +\partial a) \in P[1..i]$; or
2. $\exists r \in R_p[q]$ such that
   1. $\forall a \in A(r), (T, +\partial a) \in P[1..i]$, and
   2. $\forall C \in C(R, q) \exists s \in C$ such that either
      1. $\exists a \in A(s), (T, -\partial a) \in P[1..i]$; or
      2. $\exists t \in R_p[q]$ such that
         1. $\forall a \in A(t), (T, +\partial a) \in P[1..i]$, and
         2. $t > s$.

$+\partial.1$ looks first for a stict rule where all the antecedents are at least $\partial$-provable. If found, it is sufficient.
If $P(i+1) = (T, +\partial q)$ then either

1) $\exists r \in R_s[q] \forall a \in A(r), (T, +\partial a) \in P[1..i]$; or

2) $\exists r \in R_p[q]$ such that
   1) $\forall a \in A(r), (T, +\partial a) \in P[1..i]$, and
   2) $\forall C \in C(R, q) \exists s \in C$ such that either
      1) $\exists a \in A(s), (T, -\partial a) \in P[1..i]$; or
      2) $\exists t \in R_p[q]$ such that
         1) $\forall a \in A(t), (T, +\partial a) \in P[1..i]$, and
         2) $t > s$.

$+\partial$.2 looks first for a plausible ($\Rightarrow$) rule for $q$ where all the antecedents are at least $\partial$-provable, and seeks to show that all evidence against $q$ is defeated.
A more cautious defeasible level of proof can be defined by changing the level of proof required to eliminate counter-evidence from not $\partial$-provable to not even even supported.

$+\delta$) If $P(i + 1) = (T, +\delta q)$ then either

1) $\exists r \in R_s[q] \forall a \in A(r), (T, +\delta a) \in P[1..i]$; or

2) $\exists r \in R_p[q]$ such that

1) $\forall a \in A(r), (T, +\delta a) \in P[1..i]$, and

2) $\forall C \in C(R, q) \exists s \in C$ such that either

1) $\exists a \in A(s), (T, \neg f a) \in P[1..i]$; or

2) $\exists t \in R_p[q]$ such that

1) $\forall a \in A(t), (T, +\delta a) \in P[1..i]$, and

2) $t > s$. 
The supported $\int$ level of proof is much weaker. A formula may be both $+\int$- and $-\int$-provable at the same time. One should not behave as if a supported formula is true.

$+\int$) If $P(i + 1) = (T, +\int q)$ then either

.1) $\exists r \in R_s[q] \forall a \in A(r), (T, +\int a) \in P[1..i];$ or

.2) $\exists r \in R_p[q]$ such that

.1) $\forall a \in A(r), (T, +\int a) \in P[1..i]$, and

.2) $\forall C \in C(R, q) \exists s \in C$ such that either

.1) $\exists a \in A(s), (T, -\delta a) \in P[1..i];$ or

.2) not $(s > r)$. 

16
Implementation requirements:

1. correct;
2. traceable;
3. WWW accessible;
4. easy to modify as theoretical options are explored; and
5. efficient.

To meet these requirements, Haskell was chosen as the implementation language.

Original target was Haskell-98. Later versions have required some widely supported extensions: multi-parameter type classes.
System components:

- command line tools for: transforming descriptions to theories, provers, test theory generator.
- WWW accessible CGI tool.

The system consists of about 4000 lines of Haskell code.

A Defeasible Logic system has been developed in parallel with this one and shares some code.

The system has been tested on Solaris, Windows-9x and (with limitations) Macintosh.
Parsing components are implemented using parser combinators, for example:

```haskell
type LitName = String

data Literal = PosLit LitName
            | NegLit LitName

pLiteralP :: Parser Literal
pLiteralP = tagP "name1"
            @> (\(_,name,_) -> PosLit name)
            <|> (literalP "symbol" "~" *> tagP "name1")
            @> (\(_,name,_) -> NegLit name)
```

This coding style was used as a model for the implementation of the inference conditions.
Haskell is a pure functional language.

The monadic I/O system permits a coding style that at least looks like an imperative program.

There are restrictions. An I/O function can call any other function, but non-I/O functions can not call I/O functions.

If the prover functions are to be able to print a trace, then they must be I/O functions, but we would still like them to look purely declarative.
Data types for proving:

```haskell
data ProofStatus = Yes | No | Bottom | Pending

type ThreadedTest state = state -> IO (ProofStatus, state)

data PlusMinus = Plus | Minus

data ProofSymbol = PS_D | PS_da | PS_d | PS_S

state includes goal counters, name supplies, and a history used to avoid recomputation and loop detection.
Test combinators:

(&&&) :: ThreadedTest s -> ThreadedTest s -> ThreadedTest s
(&&&) t1 t2 s
    = do (r1,s1) <- t1 s
        case r1 of
           Yes    -> t2 s1
           No     -> return (r1, s1)
           Bottom -> return (r1, s1)
           Pending -> error "Pending in andC"
fA' :: [ThreadedTest s] -> ThreadedTest s
fA' ts s = case ts of
    []  ->
        return (Yes, s)
    [t] ->
        t s
    (t1:t2:ts) -> do
        (r1,s1) <- t1 s
        case r1 of
            Yes    -> fA' (t2:ts) s1
            No     -> return (r1,s1)
            Bottom -> return (r1,s1)
            Pending -> error "Pending in fA’"

fA :: [a] -> (a -> ThreadedTest state) -> ThreadedTest state
fA xs p = fA' $ map p xs
The $+\partial$ inference condition:

\[
\text{plus}_d\ q = \text{tE} (\text{rsq} \ q) (\lambda r \rightarrow \text{fA} (\text{ants} \ r) (\lambda a \rightarrow \text{prove} (\text{Plus} \ \text{PS}_d \ [a]))) \ ||| \\
\text{tE} (\text{rpq} \ q) (\lambda r \rightarrow \\
\text{fA} (\text{ants} \ r) (\lambda a \rightarrow \text{prove} (\text{Plus} \ \text{PS}_d \ [a]))) \ &&& \\
\text{fA} (\text{comps} \ q) (\lambda cs \rightarrow \text{tE} \ cs (\lambda c \rightarrow \text{fA} (\text{rq} \ c) (\lambda s \rightarrow \\
\text{tE} (\text{ants} \ s) (\lambda a \rightarrow \text{prove} (\text{Minus} \ \text{PS}_d \ [a]))) ||| \\
\text{tE} (\text{rpq} \ q) (\lambda t \rightarrow \\
\text{fA} (\text{ants} \ t) (\lambda a \rightarrow \text{prove} (\text{Plus} \ \text{PS}_d \ [a]))) \ &&& \\
(t \gg s) \\
) )))
\]
A description:

qu. % Nixon is a quaker.
r. % Nixon is a republican.
qu => d. % Quakers are usually doves.
r => h. % Republicans are usually hawks.
d => ~h. % Doves are usually not hawks.
h => ~d. % Hawks are usually not doves.
d => pa. % Doves and hawks are ...
h => pa. % ... usually politically active

Is Nixon politically active? This is an example where Defeasible Logic can not give a positive result.
The description is transformed into this theory:

R1: {} \rightarrow q\text{u}.
R2: {} \rightarrow r.
R3: \{q\text{u}\} \Rightarrow d.
R4: \{r\} \Rightarrow h.
R5: \{d\} \Rightarrow \neg h.
R6: \{h\} \Rightarrow \neg d.
R7: \{d\} \Rightarrow p\text{a}.
R8: \{h\} \Rightarrow p\text{a}.
R9: \{d \mid h\} \Rightarrow p\text{a}.
We find out with the query: +da pa.

To prove: (T, +da pa)
. To prove: (T, +da d)
   (11 lines deleted)
. Not proved: (T, +da d)
. To prove: (T, +da h)
   (11 lines deleted)
. Not proved: (T, +da h)
. To prove: (T, +da \{d, h\})
  . Not proved previously: (T, +da d)
  . Not proved previously: (T, +da h)
. \[ T.1 = T \cup \{\} \rightarrow \neg h \]
. . To prove: (T.1, +da d)
. . . To prove: (T.1, +da qu)
. . . Proved: (T.1, +da qu)
. . c = [R6: \{h\} \Rightarrow \neg d]
. . . To prove: (T.1, -S h)
. . . . To prove: (T.1, -S r)
. . . . Not proved: (T.1, -S r)
. . . c = [R5: \{d\} \Rightarrow \neg h]
. . . . Loop detected: (T.1, +da d)
. . . c = [\{\} \rightarrow \neg h]
. . . \{\} \rightarrow \neg h \text{ is not } \Rightarrow \text{ R4: } \{r\} \Rightarrow h
. . . c = []
. . . Proved: (T.1, -S h)
. . Proved: (T.1, +da d)
\[ T.2 = T \cup [\{} \rightarrow \neg d \] 

To prove: (T.2, +da h)

To prove: (T.2, +da r)

Proved: (T.2, +da r)

c = \{R5: \{d\} \Rightarrow \neg h\}

To prove: (T.2, -S d)

To prove: (T.2, -S qu)

Not proved: (T.2, -S qu)

c = \{R6: \{h\} \Rightarrow \neg d\}

Loop detected: (T.2, +da h)

c = [\{} \rightarrow \neg d \]

\{\} \rightarrow \neg d \text{ is not } \Rightarrow R3: \{qu\} \Rightarrow d

c = []

Proved: (T.2, -S d)

Proved: (T.2, +da h)

Proved: (T, +da \{d, h\})

Proved: (T, +da pa)
Results.

The implementation is a success as it is currently significantly aiding the theoretical development of Plausible Logic, relieving tedium and improving accuracy. New inference conditions can be added or removed within minutes.

The present definitions of the $\delta$, $-\delta$, $+\int$, and $-\int$ inference conditions were developed with the assistance of this system.

Experiments to check the performance of this implementation are encouraging.
\[
\text{levels}(n) = \begin{cases}
    r_0 : \{\} & \Rightarrow a_0 \\
    r_1 : a_1 & \Rightarrow \neg a_0 \\
    r_2 : \{\} & \Rightarrow a_1 \\
    r_3 : a_2 & \Rightarrow \neg a_1 \\
    & \quad r_3 > r_2 \\
    r_4 : \{\} & \Rightarrow a_2 \\
    r_5 : a_3 & \Rightarrow \neg a_2 \\
    r_6 : & \Rightarrow a_3 \\
    r_7 : a_4 & \Rightarrow \neg a_3 \\
    & \quad r_7 > r_6 \\
    r_8 : \{\} & \Rightarrow a_4 \\
    \quad \vdots \\
    r_{4n+1} : a_{2n+1} & \Rightarrow \neg a_{2n} \\
    r_{4n+2} : \{\} & \Rightarrow a_{2n+1} \\
    r_{4n+3} : a_{2n+2} & \Rightarrow \neg a_{2n+1} \\
    & \quad r_{4n+3} > r_{4n+2} \\
    r_{4n+4} : \{\} & \Rightarrow a_{2n+2}
\end{cases}
\]
CPU time (s) per sub-goal as a function of the number of rules in a plausible theory. The proof in each case used all of the rules and one priority for every four rules.
Array implementation performance.
For the Defeasible implementation, we can currently handle hundreds of thousands of rules. Similar improvements can be made for Plausible Logic.

For Plausible Logic, disjunction introduces exponential complexity which can not be avoided.

However in practice (business rules, regulations) the number of disjuncts is small (less than 5), or where there are more they are exclusive.
Conclusions:

• We present the first complete implementation of propositional Plausible Logic.
• The implementation is clear, and easy to extend and maintain.
• The Haskell implementation has at least adequate performance to support continuing theoretical development.

Planned future development:

• Variables.
• Pointwise disjunctions.