# Pseudo±Boolean Valued Prolog

Melvin Fitting

Department of Mathematics and Computer Science, Herbert H. Lehman College (CUNY), Bronx, New York 10468.

Department of Computer Science, The Graduate School and University Center (CUNY), 33 West 42 Street, New York, New York 10036.

Bitnet MLFLC@CUNYVM

**A generalization of conventional Horn clause logic programming is proposed in which the space of truth values is a pseudo-Boolean or Heyting algebra, whose members may be thought of as evidences for propositions. A minimal model and an operational semantics is presented, and their equivalence is proved, thus generalizing the classic work of Van Emden and Kowalski.**

## §1 Introduction.

Logic programming has generally been developed in a Classical two-valued setting; there are no "degrees of truth". What we propose here is the introduction of more general spaces of truth values, reading truth values as "evidence factors". Then conventional logic programming may be seen as the special case in which only total evidence is acceptable, and so the space of evidences degenerates to two values: evidence for; no evidence for.

A space of evidences must have some structure to it. We make an informal case for giving it the structure of a pseudo-Boolean, or Heyting algebra. This is under the restrictive assumptions that evidence is always correct, though some evidence may be better (more complete) than others; and that evidence does not change with time. Both these assumptions are unrealistic, of course, but one must begin somewhere. In a future paper we will consider further generalizations that will address these issues.

In the fundamental paper [6] an operational and a denotational (minimal model) semantics is presented for conventional Horn clause programming. Here we extend that work to Horn clause programming with a pseudo-Boolean algebra as the space of truth values. The work of [6] then becomes a special case, since the Classical two-valued space is also a pseudo-Boolean algebra.

We do not consider anything more general than Horn clause programming. However, McCarty has developed a Prolog-like system [2] in which implications may appear in clause bodies, and these are treated Intuitionistically. See also [3] and [4]. Since pseudo-Boolean algebras provide a semantics for Intuitionistic logic [5], it would be of interest to investigate a combination of evidence factors as developed here with McCarty's ideas. We do not take this up here, however.

## §2 Motivation.

Suppose we have a library $\mathbf{B}$ of books. And further, suppose the books only contain true things. Then, if we make an assertion $X$ we can justify it by citing a book $b$ in $\mathbf{B}$ that says $X$. Thus assertion-plus-evidence is a pair $\langle X, b\rangle$.

We want some structure in the library. For two books $b$ and $c$ in $\mathbf{B}$, write $b \leq c$ if every assertion in book $b$ is also in book $c$. This is a partial ordering of $\mathbf{B}$.

Further, for two books $b$ and $c$ in $\mathbf{B}$, we can always write a book containing exactly their joint assertions and add it to the library. Suppose this done. Then $\mathbf{B}$ is closed under $\cap$. Similarly we may suppose $\mathbf{B}$ closed under $\cup$. For bounds we suppose a book with blank pages, $\bot$, and a book containing the combined content of all the books in $\mathbf{B}$, $\top$.

To say $b$ and $c$ contain no common information is just to say $b \cap c = 0$. Suppose we take all books that contain no information in common with $b$, and copy all their information into a single volume, $c$. Then $c$ would be the largest (in the $\leq$ sense) book such that $b \cap c = \bot$. Call $c$ the *pseudo-complement* of $b$ and denote it by −b. Then $b \cap -b = \bot$, but not necessarily $b \cup -b = \top$.

More generally, for any $b$ and $c$ in B we may suppose a *relative* pseudo-complement exists, a largest book such that the information it has in common with $b$ is also in $c$. This is denoted $b \Rightarrow c$. Thus it is the largest volume such that $b \cap b \Rightarrow c \leq c$. Then $-b = b \Rightarrow \bot$.

As described, the library is what is known as a *pseudo-boolean algebra (PBA)*. There is a proper definition in the next section; [5] is the standard reference on their properties. Pseudo-boolean algebras are a strictly broader class than boolean algebras. The collection of all open sets of a topological space is always a PBA, but not always a boolean algebra. In particular, linear orderings give rise to PBA's, with meets and joins becoming infs and sups. $\{T, F\}$ is thus the simplest PBA. PBA's are the appropriate algebraic structure for modelling intuitionistic logic, though that is not directly involved here. What we claim is that a PBA is the right formal counterpart of the notion of a collection of evidences or reasons, partially ordered by inclusion. And what we propose is a generalization of PROLOG in which truth values constitute a complete PBA.

## §3 Pseudo-Boolean Algebras

Let $\langle \mathbf{B}, \leq \rangle$ be a lattice. For $b$ and $c$ in B, the *pseudo-complement* of $b$ relative to $c$ is the greatest element $x$ of $B$ such that $b \cap x \leq c$. It is denoted $b \Rightarrow c$ (if it exists).

A *pseudo-Boolean algebra* (PBA) is a lattice in which relative pseudo-complements always exist, and which has a least element $\bot$.

It follows that a PBA is distributive and has a greatest element, $\top$.

A PBA $\mathbf{B}$ is *complete* if $\bigcap S$ and $\bigcup S$ exist for every non-empty $S \subseteq \mathbf{B}$. In fact, it is enough to assume either closure under meets or under joins to have both.

From now on we will only be interested in complete PBA's (CPBA). Of course completeness is trivial for finite algebras.

## §4 "Conventional" Procedures

We present a PROLOG-style language with two modifications. The first, and most trivial, is that arrows go from left to right and are used exclusively, replacing conjunctions. The second is that the underlying data structure is left open. The general treatment here is based on that of [1].

*Identifiers* are $P$,$Q$,$R$,...

*Variables* are $x$,$y$,$z$,...

*Constants* are $a$,$b$,$c$,...

*Terms* are variables and constants

*Atomic formulas* are expressions of the form $P(t_1, ..., t_n)$ where $P$ is an identifier and $t_1, \ldots, t_n$ are terms.

*Formulas* are expressions of the form $A_1 \rightarrow A_2 \rightarrow A_3 \rightarrow \cdots \rightarrow A_n \rightarrow B$ where each $A_i$ and $B$ is atomic. $B$ is the *conclusion* of this formula. We think of this as parenthesized to the right. It is thus equivalent to $(A_1 \wedge A_2 \wedge A_3 \cdots \wedge A_n) \rightarrow B$ and to $A_1 \rightarrow (A_2 \rightarrow (A_3 \rightarrow \cdots (A_n \rightarrow B) \cdots))$, and these equivalences hold Intuitionistically as well as Classically. The $A_i$ list may be empty: an atomic formula itself is a formula (and is the conclusion of it).

A *data structure* is a tuple $\langle \mathbf{D}; \mathbf{R}_1, ..., \mathbf{R}_n \rangle$ in which $\mathbf{D}$ is a non-empty domain and $\mathbf{R}_1, \ldots, \mathbf{R}_n$ are relations on $\mathbf{D}$. Think of $\mathbf{R}_1, \ldots, \mathbf{R}_n$ as the "given" relations of the data structure, the facts we simply know.

We assume a data structure has been chosen and is fixed from now on. To avoid complications that are not relevant now, we assume the domain $\mathbf{D}$ is exactly the set of constant symbols $\{a, b, c, \ldots\}$. To each relation $\mathbf{R}_1, \ldots, \mathbf{R}_n$, a unique identifier $R_1$, ..., $R_n$ is associated. These identifiers are called *reserved* identifiers. A formula is called *acceptable* if no reserved identifier is used in its conclusion. The point of this restriction is to make it impossible for us to assign values to the given relations.

If we were not introducing PBA's into the picture, we would now proceed as follows.

A *procedure* $\langle S, P \rangle$ is a finite collection $S$ of acceptable formulas, together with a specified unreserved identifier $P$.

A *trace* of a procedure $\langle S, P \rangle$ is a sequence of variable–free formulas $X_1$, $X_2$, ..., such that, for each $i$, either:

1) $X_i$ is a substitution instance of a formula in $S$ (variables are replaced by constants) or

2) $X_i$ is of the form $R_j(c_1, \ldots, c_n)$ where $\langle c_1, \ldots, c_n \rangle$ is in the given relation $\mathbf{R}_j$, or

3) there are $j, k < i$ with $X_j$ atomic and $X_k$ of the form $X_j \rightarrow X_i$.

Finally, the *output* of a procedure $\langle S, P \rangle$ is the relation on $\mathbf{D}$ consisting of all tuples $\langle c_1, \ldots, c_n \rangle$ such that $P(c_1, \ldots, c_n)$ is the last line of a trace of $\langle S, P \rangle$.

We have given the "conventional" version thus far in order to contrast it with the PBA version below. We note that models can be defined and a semantics based on them developed. We leave this entirely, until the generalization later on.

## §5 Procedures with evidence factors

Let $\mathbf{B}$ be a CPBA, fixed from now on. And let the underlying programming language be as in the previous section. (Note: completeness of the pseudo-Boolean algebra is not used in this section, but it is necessary to connect things with the semantics presented in the next section.)

If $X$ is a formula and $b \in \mathbf{B}$, the pair $\langle X, b \rangle$ is a *formula with an evidence factor*. The intuitive reading is: my reason for asserting $X$ is $b$ or better. It should be noted that $\langle X, \bot \rangle$ does not mean $X$ is known to be false, but rather, that we have no evidence that $X$ is true. In fact, $\langle X, \bot \rangle$ adds nothing useful to a procedure.

A *procedure* now is $\langle S, P \rangle$ where $S$ is a set of acceptable formulas with evidence factors and $P$ is an identifier.

A *trace* of a procedure $\langle S,P\rangle$ is a sequence $\langle X_1,b_1\rangle,\langle X_2,b_2\rangle,\ldots$ where, for each $i$ either:

1) there is a member of $S$ of the form $\langle Y,b_i\rangle$ and $X_i$ is a substitution instance of $Y$, or

2) $X_i$ is of the form $R_j(t_1,\ldots,t_n)$ where $\langle t_1,\ldots,t_n\rangle\in\mathbf{R}_j$, one of the "given" relations, and $b_i=\top$, the unit of $B$, or

3) there are $j,k<i$ with $X_j$ atomic, $X_k=X_j\to X_i$, and $b_i=b_j\cap b_k$.

Item 3 amounts to adopting the rule of derivation:

$$\frac{\langle C,c\rangle\qquad\langle C\to D,d\rangle\qquad C\text{ atomic}}{\langle D,c\cap d\rangle}$$

The *output* of procedure $\langle S,P\rangle$ is the set of all items of the form $\langle P(t_1,\ldots,t_n),b\rangle$ which are last lines of traces for this procedure.

## §6 Models

We provide a kind of denotational semantics for the language of the previous section. From now on we assume we have a fixed data structure $\langle\mathbf{D};\mathbf{R}_1,\ldots,\mathbf{R}_k\rangle$ and a fixed CPBA $\mathbf{B}$.

A *valuation* is a mapping $v$ from the set of variable-free atomic formulas into $\mathbf{B}$. A valuation extends uniquely to a mapping from all formulas into $\mathbf{B}$ subject to the following conditions:

1) for $A$ atomic, $v(A\to X)=v(A)\Rightarrow v(X)$

2) for a formula $X$ with variables, $v(X)=\bigcap\{v(X')\mid X'\text{ a substitution instance of }X\}$.

We use $v$ both for the underlying valuation and for its extension to all formulas.

A valuation $v$ is *in* the data structure $\langle\mathbf{D};\mathbf{R}_1,\ldots,\mathbf{R}_k\rangle$ if, for each "given" relation $\mathbf{R}_i$, and for the identifier $R_i$ assigned to it, $v(R_i(t_1,\ldots,t_n))=\top$ iff $\langle t_1,\ldots,t_n\rangle\in\mathbf{R}_i$.

Let $S$ be a set of acceptable formulas with evidence factors. A valuation $v$ is a *model* for $S$ if $\langle X,b\rangle\in S$ implies $v(X)\geq b$.

Finally, a *model for a procedure* $\langle S,P\rangle$ is a model $v$ for $S$ in the given data structure.

Now the main results of this section are easily sketched. First, there is always a minimal model for a procedure; minimal in the sense that the truth values it assigns are $\leq$ those assigned by any other model. And second, truth values assigned by this minimal model correspond to those given by traces based on the procedure. Proper statements of these results will be found below.

Let $v_{triv}$ be the mapping such that $v_{triv}(X)=\top$ for all formulas $X$. This is a model for every procedure, hence any given procedure $\langle S,P\rangle$ always has at least one model.

Let $\langle S,P\rangle$ be a procedure. Define a mapping $v_S$ as follows. For variable–free atomic formulas $A$, $v_S(A)=\bigcap\{v(A)\mid v\text{ is a model for }\langle S,P\rangle\}$. Then $v_S$ is extended to all formulas using conditions 1) and 2) above.

*Claim 1.* For every formula $X$,
$v_S(X)\geq\bigcap\{v(X)\mid v\text{ is a model for }\langle S,P\rangle\}$.

*Proof.* We consider only one case. Say $X$ is variable-free, of the form $A\to Y$, where $A$ is atomic, and the claim has been established for simpler variable- free formulas (fewer arrows), in particular for $Y$. Then proceed as follows. Let $v_0$ be any model for $\langle S,P\rangle$. Then
$v_S(A)\cap\bigcap\{v(A\to Y)\mid v\text{ a model for }\langle S,P\rangle\}$
$\leq v_0(A)\cap v_0(A\to Y)$
$=v_0(A)\cap v_0(A)\Rightarrow v_0(Y)$
$\leq v_0(Y)$.
Since $v_0$ was an arbitrary model,
$v_S(A)\cap\bigcap\{v(A\to Y)\mid v\text{ a model for }\langle S,P\rangle\}$
$\leq\bigcap\{v(Y)\mid v\text{ a model for }\langle S,P\rangle\}$
$\leq v_S(Y)$
by induction hypothesis. But in any PBA, $b\Rightarrow c$ is the largest thing such that $b\cap b\Rightarrow c\leq c$. Hence
$\bigcap\{v(A\to Y)\mid v\text{ a model for }\langle S,P\rangle\}$
$\leq v_S(X)\Rightarrow v_S(Y)$
$=v_S(X\to Y)$.

*Claim 2.* $v_S$ is itself a model for $\langle S,P\rangle$.

*Proof.* Follows from claim 1.

$v_S$ is the minimal model for $\langle S,P\rangle$. It exactly captures the behavior of $\langle S,P\rangle$, as we proceed to show.

*Claim 3.* Let $v$ be any model for $\langle S,P\rangle$ (in particular the minimal one). If $\langle P(t_1,\ldots,t_n),b\rangle$ is an output of $\langle S,P\rangle$ then $v(P(t_1,\ldots,t_n))\geq b$.

*Proof.* Easy induction on trace length. The key step, corresponding to Rule 3 for constructing traces is handled as follows. Suppose $v(A)\geq a$ and $v(A\to Y)\geq b$. We want $v(Y)\geq a\cap b$. But
$a\cap b$
$\leq v(A)\cap v(A\to Y)$
$=v(A)\cap v(A)\Rightarrow v(Y)$
$\leq v(Y)$.

Next, define a valuation $v_{trace}$ as follows. For a variable free atomic $A$, let $v_{trace}(A)=\bigcup\{c\mid\text{some trace of }\langle S,P\rangle\text{ ends with }\langle A,c\rangle\}$. Then extend $v_{trace}$ to all formulas as usual.

*Claim 4.* For every constant-free formula $X$, $v_{trace}(X)\geq\bigcup\{c\mid\text{some trace of }\langle S,P\rangle\text{ ends in }\langle X,c\rangle\}$.

*Proof.* Suppose $X$ is $A\to Y$ where $A$ is atomic, and the result is known for $Y$ (and for $A$ by definition). Let $a_0$ be an arbitrary member of $\{a\mid$

some trace ends with $\langle A, a \rangle\}$ and let $c_0$ be an arbitrary member of $\{c \mid$ some trace ends with $\langle A \rightarrow Y, c \rangle\}$. Then certainly, using trace Rule 3, there is a trace ending with $\langle Y, a_0 \cap c_0 \rangle$. So by the induction hypothesis, $v_{trace}(Y) \geq a_0 \cap c_0$. Since $a_0$ was arbitrary, it follows that $v_{trace}(Y) \geq v_{trace}(A) \cap c_0$. But then $v_{trace}(A) \Rightarrow v_{trace}(Y) \geq c_0$, that is, $v_{trace}(A \rightarrow Y) \geq c_0$.

*Claim 5.* $v_{trace}$ is a model for $\langle S, P \rangle$.

*Proof.* If $\langle X, a \rangle \in S$, it follows from Claim 4 that $v_{trace}(X') \geq a$ for every substitution instance $X'$ of $X$, and then $v_{trace}(X) \geq a$ follows.

*Main Claim.* On "outputs" we have
$v_S(P(t_1, \ldots, t_n)) = v_{trace}(P(t_1, \ldots, t_n))$.

*Proof.* By Claim 3 we have $v_S(P(t_1, \ldots, t_n)) \geq v_{trace}(P(t_1, \ldots, t_n))$. In the other direction, by the definition of $v_S$ on variable free atomic formulas and Claim 5 it follows that
$v_S(P(t_1, \ldots, t_n)) \leq v_{trace}(P(t_1, \ldots, t_n))$.

## References.

[1] *Computability Theory, Semantics, and Logic Programming*, M. C. Fitting, Oxford University Press, New York, 1987.

[2] *Fixed Point Semantics and Tableau Proof Procedures for a Clausal Intuitionistic Logic*, L. T. McCarty, Technical Report LRP–TR–18, Rutgers University, 1986.

[3] Clausal Intuitionistic logic: an outline, L. T. McCarty, submitted to *Journal of Automated Reasoning*, 1986

[4] A Theory of modules for logic programs, D. A. Miller, *Proceedings, 1986 IEEE Symposium on Logic Programming*, 1986.

[5] *The Mathematics of Metamathematics, second edition*, H. Rasiowa and R. Sikorski, PWN – Polish Scientific Publishers, Warsaw, 1968.

[6] The Semantics of predicate logic as a programming language, M. Van Emden and R. Kowalski, *Journal of the Association for Computing Machinery*, Vol 23, pp. 733–742, 1976.

Research supported by PSC–CUNY Grants 666396, 667295 and NSF Grant CCR–8702307.

July 11, 1987