Monotonic and Residuated Logic Programs

Carlos Viegas Damásio and Luís Moniz Pereira

Centro de Inteligência Artificial (CENTRIA)
Departamento de Informática, Universidade Nova de Lisboa
2829-516 Caparica, Portugal. E-mail: `cd|lmp@di.fct.unl.pt`

Abstract. In this paper we define the rather general framework of Monotonic Logic Programs, where the main results of (definite) logic programming are validly extrapolated. Whenever defining new logic programming extensions, we can thus turn our attention to the stipulation and study of its intuitive algebraic properties within the very general setting. Then, the existence of a minimum model and of a monotonic immediate consequences operator is guaranteed, and they are related as in classical logic programming. Afterwards we study the more restricted class of residuated logic programs which is able to capture several quite distinct logic programming semantics. Namely: Generalized Annotated Logic Programs, Fuzzy Logic Programming, Hybrid Probabilistic Logic Programs, and Possibilistic Logic Programming. We provide the embedding of possibilistic logic programming.

Keywords: Logic Programming, Possibilistic Logic, Many-valued logics.

1 Introduction

The literature on logic programming theory is brimming with proposals of languages and semantics for extensions of definite logic programs (e.g. [6, 19, 4, 10]), i.e. those without non-monotonic or default negation. Usually, the authors characterize their programs with a model theoretic semantics, where a minimum model is guaranteed to exist, and a corresponding monotonic fixpoint operator (continuous or not). In many case the semantics is many-valued.

In this paper we abstract out all the details and define a rather general framework of Monotonic Logic Programs to capture the core "spirit" of logic programming. For this purpose we follow an algebraic approach to both the language and the semantics of logic programs. We were inspired by the deep theoretical results of many-valued logics and fuzzy logic (see [1, 9] for excellent accounts) and applied these ideas to logic programming. In fact, a preliminary work in this direction is [19], but the authors restrict themselves to a linearly ordered set of truth-values (the real closed interval $[0, 1]$) and to a very limited syntax: the head of a rule is a literal and the body is a multiplication (t-norm) of literals. We start by defining the notion of an implication symbol, sufficient to guarantee the validity of the standard logic programming results. Later on we resort to residuated lattices (c.f. [1, 9]), where a generalized modus ponens rule is defined. This characterizes the essence of logic programming: from the truth-value of bodies of rules for an atom we can determine the truth-value of that atom, depending on the degree of confidence in each of the rules.

Our paper proceeds as follows. In Section 2 we introduce the language of Monotonic Logic Programs and associated implication algebras. In the section after that we present our main theoretical results. Then we set forth the definitions of residual lattices and show that the Residuated Logic Programs of [2] are a special instance of Monotonic Logic Programs, where one associates with each rule a weight or confidence factor. Lastly, we show the embedding of Possibilistic Logic Programming into Residuated Logic Programs, and terminate with some conclusions and future directions.

2 Monotonic Logic Programs

The theoretical foundations of logic programming were clearly established in [11, 16] for definite logic programs (see also [12]), i.e. programs made up of rules of the form $A_0 \subset A_1 \wedge \ldots \wedge A_n (n \geq 0)$ where each $A_i (0 \leq i \leq n)$ is a propositional symbol (an atom), $\subset$ is classical implication, and $\wedge$ the usual Boolean conjunction. In this section we generalize the language of definite logic programs in order to encompass more complex bodies and heads and, evidently, many-valued logics. For simplicity, we consider only the propositional (ground) case.

When defining a (new) logic it is necessary to address the following two distinct but related aspects: the syntax and the corresponding interpretation of the logical symbols in the language. In this paper we adopt an algebraic characterization of both the language and interpretation of operators. This is a very general and powerful framework, allowing for a simple relation between the two. For lack of space, we reduce the presentation to the essentials. For more details consult for instance [8].

The main assumptions of the paper are collected in the next two definitions.

Definition 1 (Implication Algebra). *Let $\mathfrak{T} =< \mathcal{T}, \preceq >$ be a complete upper semilattice*[1] *and consider an algebra $\mathfrak{A}$ on the carrier set $\mathcal{T}$. We say that $\mathfrak{A}$ is an implication algebra with respect to $\mathfrak{T}$ iff it has defined an operator $\leftarrow$ on $\mathfrak{A}$ such that $\forall_{a_1,a_2 \in \mathfrak{T}}\ (a_1 \leftarrow a_2) = \top$ iff $a_1 \succeq a_2$ where $\top$ is the top element of $\mathfrak{T}$.*

Example 1. The closed real interval $[0, 1]$ with the usual ordering is a complete lattice. The algebra $\mathfrak{G}$ on $[0, 1]$ with Gödel implication $x \Leftarrow y = 1$ (if $x \geq y$), and $x \Leftarrow y = x$ otherwise, is an implication algebra. It is obvious that if $x < y$ then $x \Leftarrow y < 1$.

Mark that some many-valued logics have implication connectives which do not obey the property of implication algebras. We shall illustrate this in the next section.

[1] The original formulation of this definition assumed a complete lattice. As shown in [13], we can easily generalize to complete upper semilattices since the meet operation over infinite sets is never used. This is also the case for GAPs [10].

Our Monotonic and Residuated Logic Programs will be constructed from the abstract syntax induced by an implication algebra and a set of propositional symbols. The method of relating syntax and semantics in such an algebraic setting is well-known and we defer again to [8] for more details.

Definition 2 (Monotonic Logic Programs). *Let $\mathfrak{A}$ be an implication algebra with respect to a complete lattice $\mathfrak{T}$. Let Π be a set of propositional symbols and $FORM_{\mathfrak{A}}(\Pi)$ the corresponding algebra of formulae freely generated from Π and the "symbols" of operators in $\mathfrak{A}$. A monotonic logic program is a set of rules of the form $A \leftarrow \Psi$ such that:*

1. *The rule $(A \leftarrow \Psi)$ is a formula of $FORM_{\mathfrak{A}}(\Pi)$;*
2. *The head of the rule A is a propositional symbol of Π.*
3. *The body formula Ψ with propositional symbols $B_1, \ldots, B_n$ $(n \geq 0)$ corresponds to an isotonic function having those symbols as arguments.*

As usual we shall represent binary connectives in infix notation.

A rule of a monotonic logic program expresses a (monotonic) computation rule of the truth-value of the head propositional symbol from the truth-values of the symbols in the body. The monotonicity of the rule is guaranteed by the isotonicity of the function corresponding to formula Ψ: if an argument of Ψ is monotonically increased then the truth-value of Ψ also monotonically increases. The unique homomorphic extension theorem, guarantees that for every interpretation of propositional symbols there is an unique associated valuation function.

Example 2. Consider the set of propositional symbols $\Pi = \{a, b, c\}$ and the implication algebra of Example 1 with the additional operator $x \wedge y = min(x, y)$. As common, we denote the algebra by $\mathfrak{G}([0,1], \Leftarrow, \wedge)$. The arity of the operators is implicit. Mark that $\wedge$ is the meet in $[0,1]$. The formulas $b \wedge c$ and $a \Leftarrow b \wedge c$ correspond to the functions $\lambda bc. \wedge (b, c)$ and $\lambda abc. \Leftarrow (a, \wedge(b, c))$, respectively. Variables range over the interval $[0,1]$.

Mark too that we employ the same symbol to represent a connective at the syntactic level (formulas) and the corresponding operator in the underlying algebra. This simplifies presentation. A simple analysis easily concludes that $a \Leftarrow b \wedge c$ is a correct monotonic logic program rule, since $\lambda bc. \wedge (b, c) = \lambda bc.min(b, c)$ is an isotonic function.

3 Model and Fixpoint Theory for Monotonic Logic Programs

In this section we define a model and a fixpoint theory for Monotonic Logic Programs, and extend to them the classical results of logic programming. The important point to realize is that all the fundamental results of logic programming depend only on the monotonicity of the body of the rule and on the fact that it is possible to determine the truth-value of the proposition in the head from the truth-value of the rule body. Also notice that we demand all the rules to be satisfied: every implication should evaluate to $\top$.

Let us start by defining the notion of interpretation. An interpretation is simply an assignment of truth-value to each propositional symbol in the language. We assume, in the remainder of this section, an implication algebra $\mathfrak{A}$ with respect to a complete lattice $\mathfrak{T} = < \mathcal{T}, \preceq >$. The operator and implication symbol will be denoted by $\leftarrow$. Consider also that a set Π of propositional symbols is given, as well as the corresponding algebra $FORM_{\mathfrak{A}}(\Pi)$ of formulae over Π. Forthwith, the notion of interpretation is straightforward:

Definition 3 (Interpretation). *An interpretation is a mapping $I : \Pi \rightarrow \mathcal{T}$. By the unique homomorphic extension theorem, the interpretation extends uniquely to a valuation function $\hat{I} : FORM_{\mathfrak{A}}(\Pi) \rightarrow \mathcal{T}$. The set of all interpretations with respect to the implication algebra $\mathfrak{A}$ is denoted by $\mathcal{I}_{\mathfrak{A}}$.*

The ordering $\preceq$ on the truth-values in $\mathfrak{T}$ is extended to the set of interpretations as follows:

Definition 4 (Lattice of interpretations). *Consider $\mathcal{I}_{\mathfrak{A}}$ the set of all interpretations with respect to implication algebra $\mathfrak{A}$, and two arbitrary interpretations $I_1, I_2 \in \mathcal{I}_{\mathfrak{A}}$. Then, $< \mathcal{I}_{\mathfrak{A}}, \sqsubseteq >$ is a complete lattice where $I_1 \sqsubseteq I_2$ iff $\forall_{p \in \Pi}\ I_1(p) \preceq I_2(p)$. The least interpretation $\triangle$ maps every propositional symbol to the least element of $\mathcal{T}$, and the greatest interpretation $\triangledown$ maps every propositional symbol to the top element of the complete lattice of truth values $\mathcal{T}$.*

A rule of a monotonic logic program is satisfied whenever the truth-value of the rule is $\top$. A model is an interpretation which satisfies every rule in the program. Formally:

Definition 5 (Model of a program). *Consider an interpretation $I \in \mathcal{I}_{\mathfrak{A}}$. A monotonic logic program rule $A \leftarrow \Psi$ is satisfied by I iff $\hat{I}\,((A \leftarrow \Psi)) = \top$. Interpretation I is a model of a monotonic logic program P iff all rules in P are satisfied by I.*

We proceed by showing that every monotonic logic program has a least model which is the least fixpoint of a monotonic operator, along with other standard logic programming results. One such result is the immediate consequences operator, extending the results of van Emden and Kowalski [16] to the general theoretical setting of implication algebras:

Definition 6 (Immediate consequences operator). *Let P be a monotonic logic program. Define the immediate consequences operator $T_P^{\mathfrak{A}} : \mathcal{I}_{\mathfrak{A}} \rightarrow \mathcal{I}_{\mathfrak{A}}$, mapping interpretations to interpretations, as:*

$$T_P^{\mathfrak{A}}(I)(A) = lub \left\{ \hat{I}(\Psi) \text{ such that } A \leftarrow \Psi \in P \right\}$$

where A is a propositional symbol.

The immediate consequences operator evaluates the body of every rule for a propositional symbol A. The truth-value of A is simply the least upper bound of the truth-values of all the bodies of the rules for it.

Theorem 1 (Monotonicity of the immediate consequences operator). *Let I_1 and I_2 be two interpretations in $\mathcal{I}_{\mathfrak{A}}$, and P a monotonic logic program. Operator $T_P^{\mathfrak{A}}$ is monotonic, i.e. if $I_1 \sqsubseteq I_2$ then $T_P^{\mathfrak{A}}(I_1) \sqsubseteq T_P^{\mathfrak{A}}(I_2)$.*

As usual, the set of models of P is characterized by the post-fixpoints of $T_P^{\mathfrak{M}}$:

Theorem 2. *An interpretation I of $\mathcal{I}_{\mathfrak{A}}$ is a model of a monotonic logic program P iff $T_P^{\mathfrak{A}}(I) \sqsubseteq I$.*

By the Knaster-Tarski fixpoint theorem, $T_P^{\mathfrak{A}}$ has a least fixpoint. Thus:

Definition 7 (Semantics of Monotonic Logic Programs). *Let P be a monotonic logic program and M_P the least fixpoint of $T_P^{\mathfrak{A}}$. The semantics of a monotonic logic program is given by M_P, being its least model.*

Theorem 3 (Fixpoint Semantics). *Let P be a monotonic logic program, and consider the transfinite sequence of interpretations of $\mathcal{I}_{\mathfrak{A}}$:*

$$
\begin{aligned}
T_P^{\mathfrak{A}}\!\uparrow^{0} &= \Delta \\
T_P^{\mathfrak{A}}\!\uparrow^{n+1} &= T_P^{\mathfrak{A}}(T_P^{\mathfrak{M}}\!\uparrow^{n}), \ \textit{if } n+1 \textit{ is a successor ordinal} \\
T_P^{\mathfrak{A}}\!\uparrow^{\alpha} &= \textstyle\bigsqcup_{\beta<\alpha} T_P^{\mathfrak{A}}\!\uparrow^{\beta}, \ \textit{if } \alpha \textit{ is a limit ordinal}
\end{aligned}
$$

Then, there is an ordinal λ such that $T_P^{\mathfrak{A}} \uparrow^{\lambda+1} = T_P^{\mathfrak{A}} \uparrow^{\lambda}$, and the least model of P is $M_P = T_P^{\mathfrak{A}} \uparrow^{\lambda}$.

The major difference from standard classical logic programming is that our $T_P^{\mathfrak{A}}$ operator might not be continuous, and therefore more than ω iterations may be necessary to "reach" the least fixpoint. All the other important results carry over to our general framework. This possibility is unavoidable if one wants to retain generality. For the study of sufficient conditions to guarantee the continuity of the $T_P^{\mathfrak{A}}$, see [13].

We now illustrate the importance of the provisions of Definition 1 with an example:

Example 3. Reichenbach [15] devised a calculus for addressing the logical problems raised by quantum mechanics. He defined three implications, three negations, two equivalences, a conjunction and a disjunction. For our example, the three implication operators and conjunction will suffice. The set of truth-values is $\{0, 1, 2\}$ with the usual ordering. The truth-tables[2] are:

$\subset$	0	1	2
0	2	1	0
1	2	2	1
2	2	2	2

$\Leftarrow$	0	1	2
0	2	2	0
1	2	2	0
2	2	2	2

$\leftarrow$	0	1	2
0	1	1	0
1	1	1	1
2	1	1	2

$\wedge$	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

[2] For the implication connectives, the consequent truth-value appears in rows and the antecedent one in columns.

Consider the three programs below:

$$\begin{array}{lll} a \subset b \wedge c & a \Leftarrow b \wedge c & a \leftarrow b \wedge c \\ b \subset 1 & b \Leftarrow 1 & b \leftarrow 1 \\ c \subset 1 & c \Leftarrow 1 & c \leftarrow 1 \end{array}$$

The least fixpoint of the immediate consequences operator when applied to the above programs results in the same interpretation, mapping propositions a, b, and c to truth-value 1. The first implication ($\subset$) complies with the provisions of Definition 1. Therefore the interpretation so obtained is the least model of the program on the left, as can be checked easily .

For the middle program, we know that $t_1 \geq t_2$ implies $(t_1 \Leftarrow t_2) = 2$, for $t_1, t_2 \in \{0, 1, 2\}$ but the converse does not hold. In general, the least fixpoint is a model of the program but might not be minimal, as in this case: the interpretation mapping every proposition to 0 is a model.

Finally, for the program on the right, we have that $(t_1 \leftarrow t_2) = 2$ entails $t_1 \geq t_2$ but not conversely. This situation is more problematic, since the least fixpoint might not be a model of the program, as per the example. In fact, this program has no model because the implications will always be evaluated to a truth-value different from 2.

4 Residuated Logic Programs

In a non-classical setting the intended truth-value of a given rule might possibly not be absolute truth. So, a generalization of *Modus Ponens* is required to reason logically with confidence factors. In many-valued logics this issue is very well understood, namely in Fuzzy Propositional Logics [14, 1, 9]. Since one of our initial goals was to capture Fuzzy Logic Programming [5, 19], it is natural to adopt as the semantical basis the *residuated lattices* (see for instance [1]).

Definition 8 (Adjoint pair). *Let* $< P, \preceq_P >$ *be a partially ordered set and* $(\leftarrow, \otimes)$ *a pair of binary operations in* P*. We say that* $(\leftarrow, \otimes)$ *forms an adjoint pair in* $< P, \preceq_P >$ *iff:*

(a_1) *Operation* $\otimes$ *is isotonic, i.e. if* $x_1, x_2, y \in P$ *such that* $x_1 \preceq_P x_2$ *then* $(x_1 \otimes y) \preceq_P (x_2 \otimes y)$ *and* $(y \otimes x_1) \preceq_P (y \otimes x_2)$*;*

(a_2) *Operation* $\leftarrow$ *is isotonic in the first argument (the consequent) and antitonic in the second argument (the antecedent), i.e. if* $x_1, x_2, y \in P$ *such that* $x_1 \preceq_P x_2$ *then* $(x_1 \leftarrow y) \preceq_P (x_2 \leftarrow y)$ *and* $(y \leftarrow x_2) \preceq_P (y \leftarrow x_1)$*;*

(a_3) *For any* $x, y, z \in P$*, we have that* $x \preceq_P (y \leftarrow z)$ *holds if and only if* $(x \otimes z) \preceq_P y$ *holds.*

The intuition for the two above properties is immediate, the third one may be more difficult to grasp. In one direction, it is simply asserting that the following *Fuzzy Modus Ponens* rule is valid (cf. [9]):

> If x is a lower bound of $\psi \leftarrow \varphi$, and z is a lower bound of φ then a lower bound y of ψ is $x \otimes z$.

The other direction ensures that the truth-value of $y \leftarrow x$ is in fact the maximal z satisfying $x \otimes z \preceq_P y$. Mark that the implication symbol in an adjoint pair must obey the provisions of Definition 1.

Besides (a_1)–(a_3), it is necessary to assume the existence of a bottom and a top element in the lattice of truth-values, and that $\top$ is an unit element of $\otimes$. It is also usual to assume, additionally, extra conditions on the multiplication operation $(\otimes)$, namely associativity, commutativity.

Definition 9 (Residuated Lattice). *Consider the lattice $\mathfrak{L} =< L, \preceq_L>$. We say that $(\mathfrak{L}, \leftarrow, \otimes)$ is a residuated lattice whenever the following three conditions are met:*

(l_1) *$\mathfrak{L}$ is a bounded lattice: it has a bottom $(\perp)$ and a top element $(\top)$;*
(l_2) *$(\leftarrow, \otimes)$ is an adjoint pair in $\mathfrak{L}$;*
(l_3) *$(L, \otimes, \top)$ is a commutative monoid*[3].

We say that the residuated lattice is complete whenever $< L, \preceq_L>$ is complete. In this case condition (l_1) is immediately satisfied.

For residuated logic programs we resort to special implication algebra where a multiplication operation is defined, and the corresponding residuum operation (or implication), plus a constant representing the top element of the lattice of truth-values (whose set is the carrier of the algebra). Together they must define a complete residuated lattice since we intend to deal with infinite programs (theories). Obviously, a residuated algebra may have additional operators. Formally:

Definition 10 (Residuated Algebra). *Let $(\mathfrak{L}, \leftarrow, \otimes)$ be a complete residuated lattice. The implication algebra $\mathfrak{A}$ defining operators $\leftarrow, \otimes$ with respect to $\mathfrak{L}$ is a residuated algebra.*

From the example at the beginning of this section, it should be clear that in order to define the syntax of residuated logic programs it is necessary to know beforehand the underlying truth-value residuated algebra, given that each program rule must be associated with a truth-value. Thus, it is natural to generalize the language of monotonic logic programs as follows (for a particular instance see [19]):

Definition 11 (Residuated Logic Programs). *Let $\mathfrak{R}$ be a residuated algebra with respect to a residuated lattice $(\mathfrak{T}, \leftarrow, \otimes)$. A residual logic program is a monotonic logic program over $\mathfrak{R}$ with rules of the form $A \leftarrow \vartheta \otimes \Psi$ where ϑ is a truth-value in $\mathfrak{T}$, and Ψ an arbitrary isotonic formula. A (weighted) rule $A \leftarrow \vartheta \otimes \Psi$ is represented*[4] *by $A \overset{\vartheta}{\leftarrow} \Psi$.*

[3] It is shown in [13] that (l_3) can be substituted by the weaker condition $\top \otimes \vartheta = \vartheta \otimes \top = \vartheta$.

[4] We are assuming that every truth-value has a corresponding constant mapping in $\mathfrak{R}$.

Consequently, residuated logic programs are a special class of monotonic ones. The important result is the following:

Theorem 4 (Model of a Residuated Logic Program). *An interpretation $I \in \mathcal{I}_{\mathfrak{R}}$ is a model of a residuated logic program P iff $\hat{I}\,(A \leftarrow \Psi) \succeq \vartheta$ for every weighted rule $A \overset{\vartheta}{\leftarrow} \Psi$ in P.*

Therefore, all the theorems of monotonic logic programs carry over directly to residuated ones, under the notion of model portrayed in the above theorem. In [13] several implication operators can be put to use. However, it is easily seen that the embedding of Definition 11 can be trivially adapted to handle several adjoint pairs in the lattice of truth-values, and so the corresponding notion of model is the same. Thus Multi-Adjoint Logic Programs are indeed an instance of Monotonic ones.

5 Possibilistic Logic Programming

As summarized in [7], possibilistic logic is a logic of uncertainty tailored for reasoning under incomplete evidence and partially inconsistent knowledge. Each formula in the language is assigned a weight in a totally ordered set, corresponding to a lower bound on the degree of necessity or possibility of the formula. The degree of necessity of a formula states to what extent the available evidence entails the truth of the formula, while the degree of possibility expresses to what extent the truth of a formula is not incompatible with the available evidence. The theory is built upon the notion of possibility measure Π [20], obeying the following axioms:

$$\Pi(\bot) = 0 \quad \Pi(\top) = 1 \quad \forall_{p,q}\ \Pi(p \vee q) = max(\Pi(p), \Pi(q))$$

The logic is not truth-functional since in general it is only guaranteed that $\Pi(p \wedge q) \leq \min(\Pi(p), \Pi(q))$. We base our presentation on [6] and consider as well only propositional clauses. For the full theory the reader is referred to the excellent overview of [7].

A propositional possibilistic clause is either a pair $(c\ (N\alpha))$ or a pair $(c\ (\Pi\beta))$, where c is a propositional clause, $\alpha \in\ ]0,1]$ and $\beta \in [0,1]$. In general terms, the semantics is engendered from the possibility measures which satisfy the set of possibilistic clauses:

- The formula $(c\ (N\alpha))$ states that c is certain at least to degree α, i.e. $N(c) \geq \alpha$ which is equivalent to $\Pi(\neg c) \leq 1 - \alpha$.
- The expression $(c\ (\Pi\beta))$ states that c is possible in some world at least to degree β, i.e. $\Pi(c) \geq \beta$.

Notice that Π and N are dual measures of necessity and possibility governed by the equation $N(c) = 1 - \Pi(\neg c)$. The truth-values $(\Pi 0), \ldots, (\Pi\beta), \ldots, (\Pi 1), \ldots, (N\alpha), \ldots, (N1)$ are totally ordered. We denote the complete lattice formed this way by $\mathcal{P}$. The semantics of a possibilistic set of clauses $\mathcal{F}$ is given by a consequence relation. We say that $(c\ w)$ is a logical consequence of $\mathcal{F}$ (denoted by $\mathcal{F} \models (c\ w)$) if every possibility measure that satisfies $\mathcal{F}$ also satisfies $(c\ w)$.

There is a formal system [7] which is sound and complete with respect to the above inconsistency-tolerant semantics of possibilistic logic. The inference rules for the propositional case are:

$$\begin{array}{ll} (GMP) & (\varphi\ w_1), (\varphi \rightarrow \psi\ w_2) \vdash (\psi\ w_1 * w_2) \\ (S) & (\varphi\ w_1) \vdash (\varphi\ w_2), \forall w_2 \leq w_1 \end{array}$$

where operation $*$ is defined by

$$\begin{array}{l} (N\alpha) * (N\beta) = (N\ \min(\alpha, \beta)) \\ (N\alpha) * (\Pi\beta) = \begin{cases} (\Pi\beta) \text{ if } \alpha + \beta > 1 \\ (\Pi 0) \text{ if } \alpha + \beta \leq 1 \end{cases} \\ (\Pi\alpha) * (\Pi\beta) = (\Pi 0) \end{array}$$

The important point for our discussion is that $*$ is a multiplication operation and therefore we can define an appropriate residuum operator ($\twoheadleftarrow$) such that jointly they form an adjoint pair. In our setting, and given an interpretation I such that $\hat{I}(\varphi) = w_1$ we know that

$$\hat{I}(\psi \twoheadleftarrow \varphi) \geq w_2 \text{ iff } \hat{I}(\psi) \geq \hat{I}(\varphi) * w_2 \text{ iff } \hat{I}(\psi) \geq w_1 * w_2$$

similarly to the above inference rule (GMP). Notice that our interpretations might not be models of the possibilistic theory. However, we are able to extract some information regarding the possibility or necessity degree of propositional symbols. This is a technique quite similar to the one employed by Kifer and Subrahmanian [10] for embedding van Emden's quantitative rules into GAPs [17].

To simplify the presentation we map the truth values in $\mathcal{P}$ to the real interval $[0, 1]$, as follows: truth-value $(\Pi\ \beta)$ is mapped to $\frac{\beta}{2}$ and truth-value $(N\ \alpha)$ corresponds to $\frac{1+\alpha}{2}$. This is a bijection. The multiplication operator $*$ is isomorphic to operator $\times$ in the definition next:

Definition 12 (Possibilistic Residuated Algebra). *Let $\mathfrak{P}$ be the residuated algebra on $[0, 1]$ with multiplication $\times$ and implication $\twoheadleftarrow$ operators defined by:*

$$p_1 \times p_2 = \begin{cases} 0 & \text{if } p_1 + p_2 \leq 1 \\ \min(p_1, p_2) & \text{if } p_1 + p_2 > 1 \end{cases}$$

$$p_1 \twoheadleftarrow p_2 = \begin{cases} 1 & \text{if } p_1 \geq p_2 \\ p_1 & \text{if } p_1 < p_2 \text{ and } p_1 + p_2 > 1 \\ 1 - p_2 & \text{if } p_1 < p_2 \text{ and } p_1 + p_2 \leq 1 \end{cases}$$

A Possibilistic Logic Program [6] is a finite set of (first-order) possibilistic Horn Clauses annotated only with necessity degrees. We consider just the case of propositional Possibilistic Logic Programs. We have the following expected result:

Theorem 5. *Let $\mathcal{F}$ be a propositional possibilistic logic program, a set of possibilistic Horn clauses $(B_1 \wedge \ldots \wedge B_n \ \rightarrow A \ w)$ where $A, B_1, \ldots B_n$ are propositional symbols and w is a weight of the form $(N\alpha)$. We construct monotonic logic program P over $\mathfrak{P}$ by translating each possibilistic Horn clause to the attendant monotonic logic program rule $A \twoheadleftarrow p \times B_1 \times \ldots \times B_n$ where $p = \frac{1+\alpha}{2}$ is the real number in $[0,1]$ corresponding to weight w. Then $\mathcal{F} \models (A \ w)$ iff $w \leq (\mathrm{lfp}\ T_P^{\mathfrak{P}})(A)$.*

For the case of possibilistic Horn clauses annotated with possibility degrees which is not addressed in [6], our own translation is sound but not complete, as discussed in the next example:

Example 4. Consider the possibilistic theory:

$$(a \wedge b \rightarrow \ q(N1)) \quad (p \rightarrow a \ (N1)) \quad (p \rightarrow b \ (N1)) \quad (p \ (\Pi 1))$$

Translating the above to a residual logic program we surmise in the least fixpoint of the immediate consequences operator that q is provable with degree $(\Pi 0)$. However, in possibilistic logic we conclude $(q \ (\Pi 1)$. The problem is that our $T_P^{\mathfrak{P}}$ operator is not aware that the proofs for $(a \ (\Pi 1)$ and $(b \ \Pi 1)$ depend on the same proposition p.

We conjecture, however, that in situations where the proof does not depend on multiple uses of the same proposition, we extract the correct conclusions:

Example 5 (adapted from [6]). Consider the knowledge base:

$$\begin{array}{l}
works(john, paris) \rightarrow lives(john, paris) \ (N \ 0.7) \\
works(mary, paris) \rightarrow lives(mary, paris) \ (N \ 0.7) \\
works(henry, paris) \rightarrow lives(henry, paris) \ (N \ 0.7) \\
lives(mary, paris) \rightarrow lives(john, paris) \ (N \ 0.6) \\
works(henry, paris) \rightarrow works(john, paris) \ (\Pi \ 0.8) \\
works(mary, paris) \ (N1) \\
works(henry, paris) \ (N1)
\end{array}$$

It translates to the monotonic logic program:

$$\begin{array}{l}
lives(john, paris) \twoheadleftarrow 0.85 \times works(john, paris) \\
lives(mary, paris) \twoheadleftarrow 0.85 \times works(mary, paris) \\
lives(henry, paris) \twoheadleftarrow 0.85 \times works(henry, paris) \\
lives(john, paris) \twoheadleftarrow 0.8 \times lives(mary, paris) \\
works(john, paris) \twoheadleftarrow 0.4 \times works(henry, paris) \\
works(mary, paris) \twoheadleftarrow 1 \\
works(henry, paris) \twoheadleftarrow 1
\end{array}$$

We gather from the least model of the above monotonic logic program that $works(mary, paris)$ and $works(henry, paris)$ have confidence 1.0, i.e. necessity degree $(N \ 1)$. Moreover, both Henry and Mary live in Paris with necessity degree $(N \ 0.7)$. Regarding John, the proposition $works(john, paris)$ is ascribed the truth-value 0.4 corresponding to possibility degree $(\Pi \ 0.8)$, and $lives(john, paris)$ receives value 0.8, meaning that it is necessary that John lives in Paris with at least degree 0.6. Such results are in accordance with Possibilistic Logic.

6 Conclusions and Further Work

The strong point of this paper is the generality of our setting, both at the language and at the semantical level. We have presented an algebraic characterization of Monotonic Logic Programs. Program rules are arbitrary monotonic body functions where the heads are propositional symbols. Our semantic structures are implication algebras where an appropriate implication operator is imposed. We then obtain a logic programming semantics with corresponding model and fixpoint theory. The major construction is a generalized immediate consequences operator, in the spirit of van Emden and Kowalski's T_p operator. The operator is monotonic and the models of a Monotonic Logic Program are its post-fixpoints. Therefore a minimum model is guaranteed to exist, it being the least fixpoint of the immediate consequences operator. Thus, when defining a new logic programming semantics we can shift attention to the stipulation and study of its intuitive semantical algebraic properties because the main results of definite logic programming carry over for free to this very general setting.

We have studied residual lattices and algebras, where a generalized form of *Modus Ponens Rule* is valid. Having defined an implication (or residuum operator), and the associated multiplication, we can assign to program rules a confidence factor, thereby defining residuated logic programs. We show that the results of Monotonic Logic Programs carry over immediately to Residuated ones.

We provide a simple translation of Possibilistic Logic Programming into Residuated Logic Programs, and discuss the issues arising from the introduction of possibility degrees. In [2] we have previously shown that our framework can capture Hybrid Probabilistic Logic Programs [4]. Both ground Generalized Annotated Logic Programs [10] and Fuzzy Logic Programming [19] are also special cases of our setting.

The embedding of c-annotated ground GAP rules under the restricted semantics is direct, and uses a technique similar to the one presented in [2]. More interestingly, we devised a way for translating ground GAPs rules with arbitrary annotations (constant, variable, or term annotated) into a single Monotonic Logic Programming rule without occurrences of variables. This translation only assumes that (finite) meets are defined in the truth-value complete upper semi-lattice. The converse is also possible, using an extension of the embedding of Fuzzy Logic Programs into Annotated Logic Programming [18]. It shall be the subject of forthcoming work.

Our incursion paves the way to combine and integrate several forms of reasoning into a uniform framework, namely fuzzy, probabilistic, uncertain, and paraconsistent reasoning. We have defined too [3] another class of logic programs, extending the Monotonic one, where rule bodies may be anti-monotonic functions, with well-founded and Stable Model like semantics. This brings together non-monotonic and incomplete forms of reasoning with those listed before.

Acknowledgements Work partially supported by PRAXIS projects MENTAL and FLUX. We thank V. S. Subrahmanian, J. Dix, T. Eiter, J. Alferes and L. Caires for helpful discussions. Special thanks to Manuel Ojeda-Aciego for his insightful comments and for sending us a preliminary version of the paper [13]. We also thank Peter Vojtás for letting us know of the existence of [18].

References

1. L. Bolc and P. Borowik. *Many-Valued Logics. Theoretical Foundations.* Springer–Verlag, 1992.
2. C. V. Damásio and L. M. Pereira. Hybrid probabilistic logic programs as residuated logic programs. In M. O. Aciego, I. P. de Guzmán, G. Brewka, and L. M. Pereira, editors, *Logics in AI, Proceedings of JELIA'00*, pages 57–72. LNAI 1919, Springer–Verlag, September 2000.
3. C. V. Damásio and L. M. Pereira. Antitonic logic programs. In *Proc. LPNMR'01*, September 2001. To appear.
4. A. Dekhtyar and V. S. Subrahmanian. Hybrid probabilistic programs. *Journal of Logic Programming*, 43(3):187–250, 2000.
5. D. Dubois, J. Lang, and H. Prade. Fuzzy sets in approximate reasoning, part 2: Logical approaches. *Fuzzy Sets and Systems*, 40:203–244, 1991.
6. D. Dubois, J. Lang, and H. Prade. Towards possibilistic logic programming. In *Proc. of ICLP'91*, pages 581–598. MIT Press, 1991.
7. D. Dubois, J. Lang, and H. Prade. Possibilistic logic. In D.M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 3, pages 439–513. Oxford University Press, 1994.
8. J. H. Gallier. *Logic for Computer Science.* John Wiley & Sons, 1987.
9. P. Hájek. *Metamathematics of Fuzzy Logic.* Trends in Logic. Studia Logica Library. Kluwer Academic Publishers, 1998.
10. M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *J. of Logic Programming*, 12:335–367, 1992.
11. R. Kowalski. Predicate logic as a programming language. In *Proceedings of IFIP'74*, pages 569–574. North Holland Publishing Company, 1974.
12. J. W. Lloyd. *Foundations of Logic Programming.* Springer–Verlag, 1987. 2nd. edition.
13. J. Medina, M. Ojeda-Aciego, and P. Vojtas. Multi-adjoint logic programming with continuous semantics. In *Proc. LPNMR'01*, September 2001. To appear.
14. J. Pavelka. On fuzzy logic I, II, III. *Zeitschr. f. Math. Logik und Grundl. der Math.*, 25, 1979.
15. H. Reichenbach. Reply to Ernest Nagel's criticism of my views on quantum mechanics. *Journal of Philosophy*, 43:239–247, 1946.
16. M. van Emden and R. Kowalski. The semantics of predicate logic as a programming language. *Journal of ACM*, 4(23):733–742, 1976.
17. M. H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 3(1):37–53, 1986.
18. P. Vojtas. Annotated and fuzzy logic programs - relationship and comparison of expressive power. Technical report, Safárik University, Slovakia, 2001.
19. P. Vojtás and L. Paulík. Soundness and completeness of non-classical extended SLD-resolution. In *Proc. of the Ws. on Extensions of Logic Programming (ELP'96)*, pages 289–301. LNCS 1050, Springer–Verlag., 1996.
20. L. A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1:3–28, 1978.