A Completeness Theorem for Multi-Adjoint Logic Programming

Jesús Medina,[1] Manuel Ojeda-Aciego,[1]* and Peter Vojtáš[2]

[1] Dept. Matemática Aplicada. Universidad de Málaga.***
{jmedina,aciego}@ctima.uma.es
[2] Mathematical Institute. Slovak Academy of Science.†
vojtas@kosice.upjs.sk

Abstract. Multi-adjoint logic programs generalise monotonic and residuated logic programs [2] in that simultaneous use of several implications in the rules and rather general connectives in the bodies are allowed. As our approach has continuous fixpoint semantics, in this work, a procedural semantics is given for the paradigm of multi-adjoint logic programming and a completeness result is proved.
Some applications which could benefit from this theoretical approach have been commented on, such as threshold computation, fuzzy databases and general fuzzy resolution.

Keywords: Approximate reasoning, fuzzy logic programming, completeness results, continuous fix-point semantics.

Technical Area T5: Mathematics.

1 Introduction

Several papers can be found in the literature on applications of definite fuzzy logic programming, which are based either on Łukasiewicz, or product, or Gödel implications on the unit real interval (an overview can be seen in [10]); if one is interested in more complex systems, it is reasonable to allow room for several different implications, and considering more general sets of truth-values to reflect uncertainty.

For instance, in [9] a system is presented where connectives are learnt from different users' examples and, therefore, one could imagine such a scenario in which a knowledge base is described by a many valued logic program where connectives have many valued truth functions and @ is an aggregation operator (e.g. arithmetic mean, weighted sum, ...).

$$\begin{aligned}\texttt{hotel\_reservation(Business\_Location, Time, Hotel)} \longleftarrow_i &\\ @(\texttt{near\_to}(\texttt{Business\_Location},\texttt{Hotel}),&\\ \texttt{cost\_reasonable}(\texttt{Hotel},\texttt{Time}),&\\ \texttt{building\_is\_fine}(\texttt{Hotel})).\ \texttt{with truth value}\ 0.8&\end{aligned}$$

where different implications could be needed for different purposes.

The following example is taken, and simplified, from [3] which shows the usefulness of considering truth-values in set other than the unit interval: Assume that if the CEO of a company sells the stock, retires with a probability over 85%, then the probability that the stock of the company drops is 40-90%

$$\texttt{price\_drop} : [0.4, 0.9] \leftarrow (\texttt{ch\_sells\_stock} \ \wedge \ \texttt{ch\_retires}) : [0.85, 1]$$

Several approaches to the generalisation of the set of truth-values can be found, for instance, that given by the structure of bilattice, which has been used to handle negation in logic programming; also in [2] an extension was presented in which the set of truth-values is generalised to a

* Corresponding author. Tel: +34 952 132 871; Fax: +34 952 132 746
*** Partially supported by Spanish DGI project BFM2000-1054-C02-02
† Supported by Grant VEGA 1/7555/20

residuated lattice (in order to embed possibilistic, and hybrid probabilistic logic programs), but only one implication was considered and no study of continuity of its semantics was given.

The framework of multi-adjoint logic programming was introduced in [8] as a generalisation of the monotonic and residuated logic programs given in [2]. The special features of multi-adjoint logic programs are that (1) it is possible to use a number of different implications in the rules of our programs, (2) sufficient conditions for continuity of the immediate consequences operator are known, and (3) the requirements on the lattice of truth-values are weaken wrt the residuated approach.

In practical systems we need a computational model, whose existence is guaranteed by the continuity of the semantics. The purpose of this work is to provide a procedural semantics to the paradigm of multi-adjoint logic programming, and mathematical proofs for soundness and completeness are given.

2 Multi-adjoint structures

The main concept we introduce in this section is that of *multi-adjoint semilattice*, which will serve as a convenient generalisation of the (residuated) lattices as a representation of the set of truth-values. Extending the results in [2, 10] to a more general setting, in which different implications (Łukasiewicz, Gödel, product) and thus, several modus ponens-like inference rules are used, naturally leads to considering several *adjoint pairs* in the lattice. The formal definition is the following:

Definition 1 (Multi-Adjoint Semilattice). *Let $\langle L, \preceq \rangle$ be a complete upper semilattice (cus-lattice, for short). A* multi-adjoint semilattice *$\mathcal{L}$ is a tuple $(L, \preceq, \leftarrow_1, \&_1, \dots, \leftarrow_n, \&_n)$ satisfying the following items:*

1. *$\langle L, \preceq \rangle$ is bounded, i.e. it has bottom ($\bot$) and top ($\top$) elements;*
2. *$\top \,\&_i\, \vartheta = \vartheta \,\&_i\, \top = \vartheta$ for all $\vartheta \in L$ for $i = 1, \dots, n$;*
3. *$(\leftarrow_i, \&_i)$ is an adjoint pair in $\langle L, \preceq \rangle$ for $i = 1, \dots, n$; that is*
   (a) *Operation $\&$ is increasing in both arguments,*
   (b) *Operation $\leftarrow$ is increasing in the first argument and decreasing in the second argument,*
   (c) *For any $x, y, z \in P$, we have that $x \preceq (y \leftarrow z)$ holds if and only if $(x \,\&\, z) \preceq y$ holds.*

Remark 1. Note that residuated lattices are a special case of multi-adjoint semilattice, in which the underlying poset has a cus-lattice structure, has monoidal structure wrt $\otimes$ and $\top$, and only one adjoint pair is present.

The last property (3c) in the definition guarantees soundness of our computational model and can be adequately interpreted in terms of multiple-valued inference as asserting that the truth-value of $y \leftarrow z$ is the maximal x satisfying $x \,\&\, z \preceq_P y$, and also the validity of the following generalised modus ponens rule [5]:

> If x is a lower bound of $\psi \leftarrow \varphi$, and z is a lower bound of φ then a lower bound of ψ is $x \,\&\, z$; (in later notation $x = \vartheta$ and $z = \hat{I}(\mathcal{B})$)

From the point of view of expresiveness, it is interesting to allow extra operators to be involved with the operators in the multi-adjoint semilattice. The structure which captures this possibility is that of a **multi-adjoint $\boldsymbol{\Omega}$-algebra** which can be understood as an extension of a multi-adjoint semilattice containing a number of extra operators given by a signature Ω. For the formal definition in terms of universal algebra see [7, 8].

In practice, we will usually assume some properties on the extra operators considered. These extra operators will be assumed to be either conjunctors or disjunctors or aggregators.

Example 1. Consider $\Omega = \{\leftarrow_P, \&_P, \leftarrow_G, \&_G, \wedge_L, @\}$, the real unit interval $U = [0,1]$ with its lattice structure, and the interpretation function I defined as:

$$I(\leftarrow_P)(x,y) = \min(1, x/y) \qquad I(\&_P)(x,y) = x \cdot y$$

$$I(\leftarrow_G)(x,y) = \begin{cases} 1 & \text{if } y \le x \\ x & \text{otherwise} \end{cases} \qquad I(\&_G)(x,y) = \min(x,y)$$

$$I(@)(x,y,z) = \tfrac{1}{6}(x + 2y + 3z) \qquad I(\wedge_L)(x,y) = \max(0, x+y-1)$$

that is, connectives are interpreted as product and Gödel connectives, a weighted sum and Łukasiewicz conjunction; then $\langle U, I\rangle$ is a multi-adjoint Ω-algebra with one aggregator and one additional conjunctor (denoted $\wedge_L$ to make explicit that its adjoint implicator is not in the language).

Note that the use of aggregators as weighted sums somehow covers the approach taken in [1] when considering the evidential support logic rules of combination.

3 Syntax and Semantics of Multi-Adjoint Logic Programs

We will consider a multi-adjoint Ω-algebra $\mathfrak{L}$ whose extra operators are either conjunctors, denoted $\wedge_1, \dots, \wedge_k$, or disjunctors, denoted $\vee_1, \dots, \vee_l$, or aggregators, denoted $@_1, \dots, @_m$. (This algebra will host the computation of the truth-values of the formulas in our programs.)

Multi-adjoint logic programs will be constructed from the abstract syntax induced by a multi-adjoint algebra on a set of propositional symbols. Specifically, let Π be a set of propositional symbols and the corresponding algebra of formulas $\mathfrak{F}$ freely generated from Π by the operators in Ω. (This algebra will be used to define the syntax of our propositional language.)

Remark 2. As we are working with two Ω-algebras, and to discharge the notation, we introduce a special notation to clarify which algebra an operator belongs to, instead of continuously using either $\omega_{\mathfrak{L}}$ or $\omega_{\mathfrak{F}}$. Let ω be an operator symbol in Ω, its interpretation under $\mathfrak{L}$ is denoted $\dot{\omega}$ (a dot on the operator), whereas ω itself will denote $\omega_{\mathfrak{F}}$ when there is no risk of confusion.

The definition of multi-adjoint logic program is given as a set of rules and facts. It is important to note that our rules are *implications* in the language, and not Horn clauses. The particular syntax of these rules and facts is given below:

Definition 2 (Multi-Adjoint Logic Programs). *A* multi-adjoint logic program *is a set $\mathbb{P}$ of rules of the form $\langle (A \leftarrow_i \mathcal{B}), \vartheta\rangle$ such that:*

1. *The* rule $(A \leftarrow_i \mathcal{B})$ *is a formula of* $\mathfrak{F}$*;*
2. *The* confidence factor ϑ *is an element (a truth-value) of* L*;*
3. *The* head *of the rule* A *is a propositional symbol of* Π*.*
4. *The* body *formula* $\mathcal{B}$ *is a formula of* $\mathfrak{F}$ *built from propositional symbols* $B_1, \dots, B_n$ $(n \ge 0)$ *by the use of conjunctors* $\&_1, \dots, \&_n$ *and* $\wedge_1, \dots, \wedge_k$*, disjunctors* $\vee_1, \dots, \vee_l$ *and aggregators* $@_1, \dots, @_m$ *.*
5. Facts *are rules with body* $\top$*.*
6. *A* query *(or* goal*) is a propositional symbol intended as a question* $?A$ *prompting the system.*

Sometimes, we will represent the above pair as $A \overset{\vartheta}{\leftarrow}_i @[B_1, \dots, B_n]$, where $B_1, \dots, B_n$ are the propositional variables occurring in the body and $@$ is the aggregator obtained as a composition, which can be consider as a connective in the language.

As usual, an *interpretation* is a mapping $I\colon \Pi \to L$. The set of all interpretations of the formulas defined by the Ω-algebra $\mathfrak{F}$ in the Ω-algebra $\mathfrak{L}$ is denoted $\mathcal{I}_{\mathfrak{L}}$. Note that each of these interpretations can be uniquely extended to the whole set of formulas F_Ω.

The ordering $\preceq$ of the truth-values L can be easily extended to the set of interpretations as follows: Consider two interpretations $I_1, I_2 \in \mathcal{I}_{\mathfrak{L}}$. Then, $\langle \mathcal{I}_{\mathfrak{L}}, \sqsubseteq\rangle$ is a cus-lattice where $I_1 \sqsubseteq$

I_2 iff $I_1(p) \preceq I_2(p)$ for all $p \in \Pi$. The least interpretation $\triangle$ maps every propositional symbol to the least element $\bot$ of L.

A rule of a multi-adjoint logic program is satisfied whenever the truth-value of the rule is greater or equal than the confidence factor associated with the rule. Formally:

Definition 3 (Satisfaction, Model, Correct answer).

1. *An interpretation $I \in \mathcal{I}_{\mathfrak{L}}$* satisfies *a weighted rule $\langle A \leftarrow_i \mathcal{B}, \vartheta\rangle$ iff $\vartheta \preceq \hat{I}\,(A \leftarrow_i \mathcal{B})$.*
2. *An interpretation $I \in \mathcal{I}_{\mathfrak{L}}$ is a* model *of a multi-adjoint logic program $\mathbb{P}$ iff all weighted rules in $\mathbb{P}$ are satisfied by I.*
3. *An element $\lambda \in L$ is a* correct answer *for a program $\mathbb{P}$ and a query ?A if for any interpretation $I \in \mathcal{I}_{\mathfrak{L}}$ which is a model of $\mathbb{P}$ we have $\lambda \preceq I(A)$.*

Note the equalities $\hat{I}(A \leftarrow_i \mathcal{B}) = \hat{I}(A) \dot{\leftarrow}_i \hat{I}(\mathcal{B}) = I(A) \dot{\leftarrow}_i \hat{I}(\mathcal{B})$ and the evaluation of $\hat{I}(\mathcal{B})$ proceeds inductively as usual, till all propositional symbols in $\mathcal{B}$ are reached and evaluated under I. For the particular case of a fact (a rule with $\top$ in the body) satisfaction of $\langle A \leftarrow_i \top, \vartheta\rangle$ means $\vartheta \preceq \hat{I}(A \leftarrow_i \top) = I(A) \dot{\leftarrow}_i \top$ by property (3c) in Def. 1 this is equivalent to $\vartheta \mathbin{\dot{\&}_i} \top \preceq I(A)$ and this, by assumption (2) in Def. 1, gives $\vartheta \preceq I(A)$.

4 Fix-point semantics and continuity

The main concepts and results here about fixpoint semantics and continuity are taken from [8].

The immediate consequences operator, given by van Emden and Kowalski in [4], can be generalised to the framework of multi-adjoint logic programs as follows:

Definition 4. *Let $\mathbb{P}$ be a multi-adjoint logic program. The* immediate consequences operator *$T_{\mathbb{P}}^{\mathfrak{L}}\colon \mathcal{I}_{\mathfrak{L}} \to \mathcal{I}_{\mathfrak{L}}$, mapping interpretations to interpretations, is defined by considering*

$$T_{\mathbb{P}}^{\mathfrak{L}}(I)(A) = \sup\left\{\vartheta \mathbin{\dot{\&}_i} \hat{I}(\mathcal{B}) \mid A \overset{\vartheta}{\leftarrow}_i \mathcal{B} \in \mathbb{P}\right\}$$

All the suprema involved in the definition do exist because L is assumed to be a cus-lattice. The monotonicity of the operator $T_{\mathbb{P}}^{\mathfrak{L}}$, and the semantics of a multi-adjoint logic program, characterised by the least fixpoint of $T_{\mathbb{P}}^{\mathfrak{L}}$, can be found in [8].

Recall that the fixpoint theorem works even without any further assumptions on conjunctors (definitely they need not be commutative and associative).

A first result in this approach is that whenever every operator in Ω turns out to be continuous in the lattice, then $T_{\mathbb{P}}$ is also continuous and, consequently, its least fixpoint can be obtained by a countably infinite iteration from the least interpretation.

The definition of continuous function which will be used.

Definition 5.

1. *Let L be a complete upper semilattice and let $f\colon L \to L$ be a mapping. We say that f is* continuous *if it preserves suprema of directed sets, that is, given a directed set X one has*

$$f(\sup X) = \sup\{f(x) \mid x \in X\}$$

   A mapping $g\colon L^n \to L$ is said to be continuous *provided that it is continuous in each argument separately.*
2. *Let $\mathfrak{F}$ be a language interpreted on a multi-adjoint Ω-algebra $\mathfrak{L}$, and let ω be any operator symbol in the language. We say that ω is* continuous *if its interpretation under $\mathfrak{L}$, that is $\dot{\omega}$, is continuous in L.*

Theorem 1 ([8]).

1. *If all the operators occurring in the bodies of the rules of a program $\mathbb{P}$ are continuous, and the adjoint conjunctions are continuous in their second argument, then $T_{\mathbb{P}}^{\mathfrak{L}}$ is continuous.*

2. *If the operator $T_{\mathbb{P}}^{\mathfrak{L}}$ is continuous for all program $\mathbb{P}$ on $\mathfrak{L}$, then any operator in the body of the rules is continuous.*

It is possible to generalise the previous theorem by requiring weaker continuity conditions on the operators but, at the same time, restricting the structure of the set of truth-values. This topic is treated in detail in [8].

5 Procedural semantics of multi-adjoint logic programming

When facing a practical system we need a computational model, which is sound and complete; the existence of such a computational model is guaranteed by the fact that the least fixpoint (model) can be attained in at most countably many steps. A sufficient condition for this at most countably iteration is given by the continuity of the semantics.

Once we have shown that the $T_{\mathbb{P}}^{\mathfrak{L}}$ operator can be continuous under very general hypotheses, then the least model can be reached in at most countably many iterations. Therefore, it is worth to define a procedural semantics which allow us to actually construct the answer to a query against a given program.

In this section we provide a procedural semantics to the paradigm of multi-adjoint logic programming, and mathematical proofs for soundness and completeness are given. Proofs are not included in this paper due to space restriction, the interested reader can get them in [7].

In the following, we will be working in a hybrid Ω-algebra made up from the elements of the lattice, and the same alphabet of the language without the adjoint implicators.

For the formal description of the computational model, we will consider an extended language $\mathfrak{F}$ defined on the same ranked set, but whose carrier is the disjoint union $\Pi \cup L$; this way we can work simultaneously with propositional symbols and with the truth-values they represent.

Definition 6 (Extended language). *Let $\mathbb{P}$ be a multi-adjoint logic program on a multi-adjoint Ω-algebra $\mathfrak{L}$ with carrier L and let V be the set of truth values of the rules in $\mathbb{P}$. The* extended language *$\mathfrak{F}^e$ is the corresponding Ω^--algebra of formulas freely generated from the disjoint union of Π and V, where Ω^- is the signature Ω without the implication connectives.*

We will refer to the formulas in the language $\mathfrak{F}^e$ simply as *extended formulas,* or *e-formulas.* An operator symbol ω interpreted under $\mathfrak{F}^e$ will be denoted as $\bar{\omega}$.

Our computational model will take a query (an atom), and will provide a lower bound of the value of A under any model of the program. Intuitively, the computation proceeds by, somehow, substituting propositional symbols by lower bounds of their truth-value until, eventually, an extended formula with no propositional symbol is obtained, which will be interpreted in the multi-adjoint semilattice to get the computed answer.

Given a program $\mathbb{P}$, we define the following admissible rules for transforming any e-formula.

Definition 7. Admissible rules *are defined as follows:*

1. *Substitute an atom A in an e-formula by $(\vartheta \bar{\&}_i \mathcal{B})$ whenever there exists a rule $\langle A \leftarrow_i \mathcal{B}, \vartheta \rangle$ in $\mathbb{P}$.*
2. *Substitute an atom A in an e-formula by $\bot$ (just to cope with unsuccesful branches).*
3. *Substitute an atom A in an e-formula by ϑ whenever there exists a fact $\langle A \leftarrow_i \top, \vartheta \rangle$ in $\mathbb{P}$.*

Note that if an e-formula turns out to have no propositional symbols, then it can be directly interpreted in the multi-adjoint Ω-algebra $\mathfrak{L}$. This justifies the definition of *computed answer.*

Definition 8 (Computed Answer). *Let $\mathbb{P}$ be a program in a multi-adjoint language interpreted on a multi-adjoint semilattice $\mathfrak{L}$ and let $?A$ be a goal. An element $\dot{@}[r_1, \dots, r_m]$, with $r_i \in L$, for all $i \in \{1, \dots, m\}$ is said to be a* computed answer *if there is a sequence $G_0, \dots, G_{n+1}$ such that*

1. *$G_0 = A$ and $G_{n+1} = \bar{@}[r_1, \dots, r_m]$ where $r_i \in L$ for all $i = 1, \dots n$.*
2. *Every G_i, for $i = 1, \dots, n$, is a formula in $\mathfrak{F}^e$.*
3. *Every G_{i+1} is inferred from G_i by one of the admissible rules.*

Remark 3. The idea of the computation is to consecutively apply admissible rules until an extended formula with no propositional symbols $\bar{@}[r_1, \dots, r_m]$ is obtained, which can be interpreted as the element $\dot{@}[r_1, \dots, r_m]$ in the lattice $\mathcal{L}$.

Note that every computed answer is correct, this is guaranteed by the property (3c) of multi-adjoint semilattice).

5.1 Reductants

It might be the case that for some lattices it is not possible to get the correct answer, for instance when we have a finite number of rules $A \overset{\vartheta_i}{\longleftarrow}_i @_i(D^i_1, \dots, D^i_{n_i})$ for $i = 1, \dots, k$, and we are working with a non-linear lattice of truth-values.

As any rule $A \overset{\vartheta_i}{\longleftarrow}_i @_i(D^i_1, \dots, D^i_{n_i})$ contributes with the value $\vartheta_i \mathbin{\dot{\&}_i} b_i$ for the calculation of the lower bound for the truth-value of A, we would like to have the possibility of reaching the supremum of all the contributions, in the computational model, in a single step. In order a single rule computes this supremum then the following *reductant property* has to be fulfilled (which is a generalisation of the concept of reductant [6]):

For the multiset $\&_1, \dots, \&_k$ of residual conjunctors of the implications $\leftarrow_i$, and their corresponding confidence values $\vartheta_1, \dots, \vartheta_k$ there exist unique $\&$, $@$ and $\vartheta \in L$ such that for any $b_1, \dots, b_k$ we have

$$\sup\{\vartheta_i \mathbin{\dot{\&}_i} b_i \mid i = 1, \dots, k\} = \vartheta \mathbin{\dot{\&}} \dot{@}(b_1, \dots, b_k)$$

If all the rules in $\mathbb{P}$ with head A are $A \overset{\vartheta_i}{\longleftarrow}_i \mathcal{B}_i$ for $i = 1, \dots, k$, then a *reductant for* A is a rule $A \overset{\vartheta}{\longleftarrow} @(\mathcal{B}_1, \dots, \mathcal{B}_n)$, where $\leftarrow$ is the residual implication of $\&$.

Provided that $\mathbb{P}$ has the reductant property, we can assume that the program contains all the reductants, since it can be easily seen that the set of models is not modified.

5.2 Completeness results

We assume here that the reductant property and the continuity of the semantics are in force. The proof of the completeness theorem follows from the fact that the least fix-point is also the least model of a program, together with the following proposition showing the behaviour of both computed and correct answers.

Proposition 1.

1. *$\lambda \in L$ is a correct answer for program $\mathbb{P}$ and query $?A$ if and only if $\lambda \preceq T^{\omega}_{\mathbb{P}}(\triangle)(A)$*
2. *Let $\mathbb{P}$ be a program, then $T^{n}_{\mathbb{P}}(\triangle)(A)$ is a computed answer for all n and for all query $?A$.*

We have now all the required background to prove a completeness result.

Theorem 2. *For every correct answer $\lambda \in L$ for a program $\mathbb{P}$ and a query $?A$, there exists a chain of elements λ_n such that $\lambda \preceq \sup \lambda_n$, such that for arbitrary n_0 there exists a computed answer δ such that $\lambda_{n_0} \preceq \delta$.*

6 Applications

In this section we briefly comment on some applications which could benefit from the theoretical developments introduced above.

One of the problems of fuzzy knowledge bases is handling a great amount of items with very small confidence value. The approach introduced in this paper enables us to propose a sound and complete threshold computation model oriented to the best correct answers up to a prescribed tolerance level, which further generalises the approach presented in [10].

The integration of information retrieval and database systems requires methods for dealing with uncertainty; this is why it is interesting to allow a multiple-valued approach to the general theory of databases. There exist already a number of such approaches, but none of them contains a formal mathematical proof of the relation between the relational algebra and its denotational counterpart. A byproduct of the procedural semantics just introduced is the possibility of defining a fuzzy relational algebra and a fuzzy Datalog. The completeness result shows that the expressive power of fuzzy Datalog is the same that the computational power of the fuzzy relational algebra.

Classical logic programming deals with Horn clauses and refutation, but this is no longer the case in a general multiple-valued framework, mainly due to the fact that $A \wedge \neg A$ can have strictly positive truth-value, but also to the fact that material implication (the truth value function of $\neg A \vee B$) has not commutative adjoint conjunctor. As our approach does not require adjoint conjunctors to be commutative, it allows the development of a sound and complete graded resolution.

7 Conclusions

We have presented a sound and complete procedural semantics for the general theory of multi-adjoint logic programming. The special features of this framework are that it is possible to use a number of different implications in the rules of our programs, sufficient conditions for continuity of the immediate consequences operator are known, and the requirements on the lattice of truth-values are weaken wrt the residuated approach.

Multi-adjoint logic program generalises monotonic and residuated logic programs [2]. Obviously, any paradigm that can be simulated by monotonic and residuated logic programs (such as either some hybrid probabilistic, or possibilistic, or non-monotonic, or fuzzy logics) can also be simulated by multi-adjoint logic programs but, in addition, some 'mixed-implications' logic paradigms can also be simulated in our approach.

Some applications which could benefit from this theoretical approach have been commented on, such as threshold computation, fuzzy databases and general fuzzy resolution.

References

1. J.F. Baldwin, T.P. Martin, and B.W. Pilsworth. *FRIL-Fuzzy and Evidential Reasoning in AI.* Research Studies Press (John Wiley), 1995.
2. C.V. Damásio and L. Moniz Pereira. Monotonic and residuated logic programs. In *Sixth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU'01.* Lect. Notes in Comp. Sci., Springer-Verlag, 2001.
3. A. Dekhtyar and V. S. Subrahmanian. Hybrid probabilistic programs. *J. of Logic Programming*, 43:187–250, 2000.
4. M. van Emden and R. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.
5. P. Hájek. *Metamathematics of Fuzzy Logic.* Trends in Logic. Kluwer Academic Publishers, 1998.
6. M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *J. of Logic Programming*, 12:335–367, 1992.
7. J. Medina, M. Ojeda-Aciego, and P. Vojtáš. A procedural semantics for multi-adjoint logic programming. Available at `http://www.satd.uma.es/aciego/TR/procsem-tr.pdf`.
8. J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Multi-adjoint logic programming with continuous semantics. In *Proc. LPNMR'01*, 2001. To appear. `http://www.satd.uma.es/aciego/TR/malp-tr.pdf`.
9. E. Naito, J. Ozawa, I. Hayashi, and N. Wakami. A proposal of a fuzzy connective with learning function. In P. Bosc and J. Kaczprzyk, editors, *Fuzziness Database Management Systems*, pages 345–364. Physica Verlag, 1995.
10. P. Vojtáš. Fuzzy logic programming. *Fuzzy sets and systems*, 2001. Accepted.