A Procedural Semantics 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] Dept. Mathematical Informatics. P.J. Šafárik University.†
vojtas@kosice.upjs.sk

Abstract. Multi-adjoint logic program generalise monotonic logic programs introduced in [1] in that simultaneous use of several implications in the rules and rather general connectives in the bodies are allowed. In this work, a procedural semantics is given for the paradigm of multi-adjoint logic programming and completeness theorems are proved.

1 Introduction

Multi-adjoint logic programming was introduced in [5] as a refinement of both initial work in [7] and residuated logic programming [1]. It allows for very general connectives in the body of the rules, and sufficient conditions for the continuity of its semantics are known. Such an approach is interesting for applications: in [6] a system is presented where connectives are learnt from different users' examples and, thus, one can imagine a scenario in which knowledge is described by a many-valued logic program where connectives have many-valued truth functions and @ is an aggregation operator (arithmetic mean, weighted sum, ...) where different implications could be needed for different purposes, and different aggregators are defined for different users, depending on their preferences, e.g. the rule below:

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

The framework of multi-adjoint logic programming was introduced in [5] as a generalisation of the monotonic and residuated logic programs given in [1]. The special features of multi-adjoint logic programs are: (1) a number of different implications are allowed in the bodies of the rules, (2) sufficient conditions for continuity of its semantics are known, and (3) the requirements on the lattice of truth-values are weaker that those for the residuated approach.

*** Partially supported by Spanish DGI project BFM2000-1054-C02-02 and Junta de Andalucía project TIC-115.
† Partially supported by Slovak project VEGA 1/7557/20

The purpose of this work is to provide a procedural semantics to the paradigm of multi-adjoint logic programming. This work is an extension of [3], with a different treatment of reductants and an improved version of the completeness results, based on the so-called supremum property.

The central topics of this paper are mainly at the theoretical level, however the obtained results can be applied in a number of contexts:

1. The integration of information retrieval and database systems requires methods for dealing with uncertainty; there are already a number of many-valued approaches to the general theory of databases, but none of them contains a formal mathematical proof of the relation between the relational algebra and its denotational counterpart; a by-product of the obtained results is the possibility of defining a fuzzy relational algebra and a fuzzy Datalog, the completeness result then shows that the expressive power of fuzzy Datalog is the same that the computational power of the fuzzy relational algebra;
2. 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.
3. The multi-adjoint framework can also be applied to abduction problems. In [4] the possibility of obtaining the cheapest possible explanation to an abduction problem wrt a cost function by means of a logic programming computation followed by a linear programming optimization has been shown.

2 Preliminary definitions

To make this paper as self-contained as possible, the necessary definitions about multi-adjoint structured are included in this section. For motivating comments, the interested reader is referred to [5].

Definition 1. *Let $\langle L, \preceq\rangle$ be a complete lattice. A* multi-adjoint lattice $\mathcal{L}$ *is a tuple $(L, \preceq, \leftarrow_1, \&_1, \ldots, \leftarrow_n, \&_n)$ satisfying the following items:*

1. *$\langle L, \preceq\rangle$ is bounded, i.e. it has bottom and top elements;*
2. *$\top \,\&_i\, \vartheta = \vartheta \,\&_i\, \top = \vartheta$ for all $\vartheta \in L$ for $i = 1, \ldots, n$;*
3. *$(\leftarrow_i, \&_i)$ is an adjoint pair in $\langle L, \preceq\rangle$ for $i = 1, \ldots, n$; i.e.*
   (a) *Operation $\&_1$ is increasing in both arguments,*
   (b) *Operation $\leftarrow_i$ is increasing in the first argument and decreasing in the second argument,*
   (c) *For any $x, y, z \in P$, we have that $x \preceq (y \leftarrow_i z)$ holds if and only if $(x \,\&_i\, z) \preceq y$ holds.*

From the point of view of expressiveness, it is interesting to allow extra operators to be involved with the operators in the multi-adjoint lattice. The structure which captures this possibility is that of a *multi-adjoint Ω-algebra* which can be understood as an extension of a multi-adjoint lattice containing a number of extra operators given by a signature Ω.

We will be working with two Ω-algebras: the first one, $\mathfrak{F}$, to define the syntax of our programs, and the second one, $\mathfrak{L}$, to host the manipulation of the truth-values of the formulas in the programs. To avoid possible name-clashes, we will denote the interpretation of an operator symbol ω in Ω under $\mathfrak{L}$ as $\dot{\omega}$ (a dot on the operator), whereas ω itself will denote its interpretation under $\mathfrak{F}$.

Definition 2 (Multi-Adjoint Logic Programs). *A* multi-adjoint logic program on a multi-adjoint Ω-algebra $\mathfrak{F}$ with values in a multi-adjoint lattice $\mathfrak{L}$ *(in short* multi-adjoint program*) is a set $\mathbb{P}$ of rules of the form $\langle (A \leftarrow_i \mathcal{B}), \vartheta \rangle$.*

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 $B_1, \ldots, B_n$ $(n \geq 0)$ by the use of conjunctors $\&_1, \ldots, \&_n$ and $\wedge_1, \ldots, \wedge_k$, disjunctors $\vee_1, \ldots, \vee_l$ and aggregators $@_1, \ldots, @_m$.*
5. Facts *are rules with body $\top$.*
6. *A* query *(or* goal*) is a propositional symbol intended as a question $?A$ prompting the system.*

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, $\hat{I} \colon F_\Omega \to L$.

The ordering $\preceq$ of the truth-values L can be easily extended to $\mathcal{I}_{\mathfrak{L}}$, which also inherits the structure of complete lattice.

Definition 3.

1. *An interpretation $I \in \mathcal{I}_{\mathfrak{L}}$* satisfies *$\langle A \leftarrow_i \mathcal{B}, \vartheta \rangle$ if and only if $\vartheta \preceq \hat{I}\,(A \leftarrow_i \mathcal{B})$.*
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 $\lambda \preceq I(A)$.*

The immediate consequences operator, given by van Emden and Kowalski, can be easily generalised to the framework of multi-adjoint logic programs.

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

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

As usual, the semantics of a multi-adjoint logic program is characterised by the post-fixpoints of $T^{\mathfrak{L}}_{\mathbb{P}}$, see [5]; that is, an interpretation I of $\mathcal{I}_{\mathfrak{L}}$ is a model of a multi-adjoint logic program $\mathbb{P}$ iff $T^{\mathfrak{L}}_{\mathbb{P}}(I) \sqsubseteq I$. It is remarkable the fact that this result is still true even without any further assumptions on conjunctors (definitely they need not be commutative and associative).

Regarding continuity, the following theorem was proved in [5].

Theorem 1 ([5]).

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.*

3 Procedural semantics of multi-adjoint logic programs

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.

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

Definition 5. *Let $\mathbb{P}$ be a multi-adjoint program, and let $V \subset L$ be the set of truth values of the rules in $\mathbb{P}$. The* extended language $\mathfrak{F}'$ *is the corresponding Ω-algebra of formulas freely generated from the disjoint union of Π and V.*

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

Our computational model will take a query (atom), and will provide a lower bound of the value of A under any model of the program. Given a program $\mathbb{P}$, we define the following admissible rules for transforming any extended formula.

Definition 6. Admissible rules *are defined as follows:*

1. *Substitute an atom A in an extended 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 extended formula by $\bot$.*
3. *Substitute an atom A in an extended formula by ϑ whenever there exists a fact $\langle A \leftarrow_i \top, \vartheta \rangle$ in $\mathbb{P}$.*

Note that if an extended 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 following definition of *computed answer.*

Definition 7. *Let $\mathbb{P}$ be a multi-adjoint program, and let $?A$ be a goal. An element $\dot{@}[r_1, \ldots, r_m]$, with $r_i \in L$, for all $i \in \{1, \ldots, m\}$ is said to be a* computed answer *if there is a sequence $G_0, \ldots, G_{n+1}$ such that*

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

Note that our procedural semantics, instead of being refutation-based (this is not possible, since negation is not allowed in our approach), is oriented to obtaining a bound of the optimal correct answer of the query.

3.1 Reductants

It might be the case that for some lattices it is not possible to get the correct answer, simply consider L to be the powerset of a two-element set $\{a, b\}$ ordered by inclusion, and the following example from Morishita, used in [2]:

Example 1. Given a multi-adjoint program $\mathbb{P}$ with rules $A \overset{a}{\leftarrow} B$ and $A \overset{b}{\leftarrow} B$ and fact $(B, \top)$. Assuming that the adjoint conjunction to $\leftarrow$ has the usual boundary conditions, then the correct answer to the query $?A$ is $\top$, since it has to be an upper bound of all the models of the program, therefore it has to be greater than both a and b. But the only computed answers are either a or b. $\square$

The idea to cope with this problem is the generalisation of the concept of reductant [2] to our framework. Namely, that whenever we have a finite number of rules $A \overset{\vartheta_i}{\leftarrow}_i @_i(D^i_1, \ldots, D^i_{n_i})$ for $i = 1, \ldots, k$, then there should exist another rule which allows us to get the correct value of A under the program. That can be rephrased as follows:

As any rule $A \overset{\vartheta_i}{\leftarrow}_i @_i(D_1, \ldots, D_{n_i})$ contributes with the value $\vartheta_i \,\dot{\&}_i\, b_i$ 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. This leads to the following definition.

Definition 8 (Reductant). *Let $\mathbb{P}$ be a multi-adjoint program; assume that all the rules in $\mathbb{P}$ with head A are $A \overset{\vartheta_i}{\leftarrow}_i \mathcal{B}_i$ for $i = 1, \ldots, n$. A* reductant for A *is a rule $A \overset{\vartheta}{\leftarrow} @(\mathcal{B}_1, \ldots, \mathcal{B}_n)$ such that for any $b_1, \ldots, b_n$ we have*

$$\sup\{\vartheta_i \,\dot{\&}_i\, b_i \mid i = 1, \ldots, n\} = \vartheta \,\dot{\&}\, \dot{@}(b_1, \ldots, b_n)$$

where $\&$ is the adjoint conjunctor to the implication $\leftarrow$.

Remark 1. When all the elements in the multiset are the same (e.g. all rules in the program have the same implication), and the operator $\dot{\&}$ is continuous, then the following equality holds

$$\sup\{\vartheta_i \,\dot{\&}\, b_i \mid i = 1, \ldots, n\} = \sup \vartheta_i \,\dot{\&}\, \sup b_i$$

which immediately implies that choosing $\vartheta = \sup \vartheta_i$ and $\dot{@}(b_1, \ldots, b_n) = \sup b_i$ we have constructed a reductant.

The example below shows a program with reductants, whose truth-values range over the lattice of closed intervals in $[0, 1]$.

Example 2. Consider the lattice of all the closed intervals in $[0, 1]$, denoted $\mathcal{C}[0, 1]$ under the ordering $[a, b] \preceq [c, d]$ iff $a \leq c$ and $d \leq b$, and consider the componentwise extended definition of the Łukasiewicz, product and Gödel implications to intervals. Let $\mathbb{P}$ be the program with rules

$$\langle A \longleftarrow_P B, [\vartheta_1, \vartheta_2] \rangle \qquad \langle A \longleftarrow_G C, [\tau_1, \tau_2] \rangle$$

where $\vartheta_1 \leq \tau_1$, and facts

$$\langle B \longleftarrow_L, [\vartheta_3, \vartheta_4] \rangle \qquad \langle C \longleftarrow_P, [\vartheta_5, \vartheta_6] \rangle$$

Let us see that $\mathbb{P}$ has reductants. In this particular case, the only head in the rules is A, so we have to define a reductant for A in $\mathbb{P}$.

From the two rules, we have the associated conjunctors to the implications $\&_P, \&_G$ (also defined componentwise), given its two truth-intervals $[\vartheta_1, \vartheta_2]$ and $[\tau_1, \tau_2]$ there should exist a truth-interval $[\epsilon_1, \epsilon_2]$, a conjunctor $\&$ and an aggregator @ such that for all $[b_1, b_2], [c_1, c_2] \in \mathcal{C}[0, 1]$:

$$\sup\{[\vartheta_1, \vartheta_2] \,\&_P [b_1, b_2], [\tau_1, \tau_2] \,\&_G [c_1, c_2]\} = [\epsilon_1, \epsilon_2] \,\&\, @([b_1, b_2], [c_1, c_2])$$

In this case, it suffices to consider $\& = \&_G$, $[\epsilon_1, \epsilon_2] = [\max(\vartheta_1, \tau_1), \min(\vartheta_2, \tau_2)]$, and $@([b_1, b_2], [c_1, c_2]) = [\max(\vartheta_1 b_1, c_1), \min(\vartheta_2 b_2, c_2)]$. □

Certainly, it will be interesting to consider only programs which contain all its reductants, but this might be a too heavy condition on our programs; the following proposition shows that it is not true, therefore we can assume that a program contains all its reductants, since its set of models is not modified.

Proposition 1. *Any reductant $A \overset{\vartheta}{\leftarrow} \mathcal{B}$ of $\mathbb{P}$ is satisfied by any model of $\mathbb{P}$. In short, $\mathbb{P} \models A \overset{\vartheta}{\leftarrow} \mathcal{B}$.*

It is possible *to construct reductants* for any head-of-rule in a given program under the only requirement that the truth-value set is complete under suprema; and this is actually an assumption for all multi-adjoint programs, which are based on a multi-adjoint lattice.

Definition 9 (Construction of reductants). *Let $\mathbb{P}$ be a multi-adjoint program; assume that all the rules in $\mathbb{P}$ with head A are $\langle A \leftarrow_i \mathcal{B}_i, \vartheta_i \rangle$ for $i = 1, \ldots, n$.* A reductant for *A is any rule $\langle A \leftarrow @(\mathcal{B}_1, \ldots, \mathcal{B}_n), \top \rangle$ where $\leftarrow$ is any implication with an adjoint conjunctor (let us denote it $\&$) and the aggregator @ is defined as follows*

$$\dot{@}(b_1, \ldots, b_n) = \sup\{\vartheta_1 \,\&_1\, b_1, \ldots, \vartheta_n \,\&_n\, b_n\}$$

It is immediate to prove that the rule constructed in the definition above is actually a reductant for A in $\mathbb{P}$, and we state the fact in the following proposition.

Proposition 2. *Under the hypotheses of the previous definition, the defined rule $\langle A \leftarrow @(\mathcal{B}_1, \ldots, \mathcal{B}_n), \top \rangle$ is a reductant for A in $\mathbb{P}$.*

Note that we have followed just traditional techniques of logic programming, and discarded non-determinism by using reductants. A possible disadvantage of this technique is that the full search space must be traversed (every rule of every atom must be evaluated), although this need not be necessary in many circumstances. It is clear that some evaluation strategies might start by executing non-deterministically the rules for a given atom, and finally the reductant. This joined with some memoizing or tabling technique would not have significant overhead, and could improve performance.

3.2 Completeness results

The proof of the completeness theorems follows from some technical results. The first lemma below states that the least fix-point is also the least model of a program; the second states a characterisation of correct answers in terms of the $T_{\mathbb{P}}^{\mathfrak{L}}$ operator.

Lemma 1. *For all model I of $\mathbb{P}$ we have that $T_{\mathbb{P}}^{\omega}(\triangle) \sqsubseteq I$.*

Lemma 2. *$\lambda \in L$ is a correct answer for program $\mathbb{P}$ and query $?A$ if and only if $\lambda \preceq T_{\mathbb{P}}^{\omega}(\triangle)(A)$*

Now, in order to match correct and computed answers, the proposition below, whose proof is based induction on n, shows that any iteration of the $T_{\mathbb{P}}^{\mathfrak{L}}$ operator is, indeed, a computed answer.

Proposition 3. *Let $\mathbb{P}$ be a program, then $T_{\mathbb{P}}^{n}(\triangle)(A)$ 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 $\lambda_{n_0} \preceq \delta$.*

Proof: Consider $\lambda_n = T_{\mathbb{P}}^{n}(\triangle)(A)$. As λ is a correct answer, we have that

$$\lambda \preceq T_{\mathbb{P}}^{\omega}(\triangle)(A) = \sup\{T_{\mathbb{P}}^{n}(\triangle)(A) \mid n \in \mathbb{N}\} = \sup \lambda_n$$

since $T_{\mathbb{P}}^{\omega}(\triangle)$ is a model. Now we can choose δ to be $T_{\mathbb{P}}^{n}(\triangle)(A)$ for any $n \geq n_0$ and the theorem follows directly by the monotonicity of the $T_{\mathbb{P}}^{\mathfrak{L}}$ operator and Proposition 3. □

The theorem above can be further refined under the assumption of the so-called supremum property:

Definition 10. *A cpo L is said to satisfy the* supremum property *if for all directed set $X \subset L$ and for all ε we have that if $\varepsilon < \sup X$ then there exists $\delta \in X$ such that $\varepsilon < \delta \leq \sup X$.*

Theorem 3 below states that any correct answer can be approximated up to any lower bound.

Theorem 3. *Assume L has the supremum property, then for every correct answer $\lambda \in L$ for a program $\mathbb{P}$ and a query $?A$, and arbitrary $\varepsilon \prec \lambda$ there exists a computed answer δ such that $\varepsilon \preceq \delta$.*

Proof: As λ is a correct answer, then $\lambda \preceq T_{\mathbb{P}}^{\omega}(\triangle)(A)$, since $T_{\mathbb{P}}^{\omega}(\triangle)$ is a model of $\mathbb{P}$. By definition we have $T_{\mathbb{P}}^{\omega}(\triangle)(A) = \sup\{T_{\mathbb{P}}^{n}(\triangle)(A) \mid n \in \mathbb{N}\}$ and, by hypothesis, $\varepsilon \prec \lambda \preceq T_{\mathbb{P}}^{\omega}(\triangle)(A)$. The supremum property states that there exists an element $\delta = T_{\mathbb{P}}^{n_0}(\triangle)(A)$ in $\{T_{\mathbb{P}}^{n}(\triangle)(A) \mid n \in \mathbb{N}\}$, such that $\varepsilon \prec \delta \preceq T_{\mathbb{P}}^{\omega}(\triangle)(A)$. This finishes the proof, for $T_{\mathbb{P}}^{n_0}(\triangle)(A)$ is a computed answer, by Proposition 3. □

4 Conclusions and future work

We have presented a framework for studying more elaborate proof procedures for multi-adjoint programs: a procedural semantics has been introduced, and two quasi-completeness theorems are stated and proved.

As future work, from the theoretical side, it is necessary to further investigate lattices with the supremum property; from the not-so-theoretical side, a practical evaluation of the proposed approach has to be performed, to evaluate the appropriateness of several possible optimisation techniques.

Acknowledgements

We thank C. Damásio and L. Moniz Pereira for their interesting comments on previous versions of this work. We are also grateful to the referees who pointed out several weaknesses of the first draft of the paper.

References

1. 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*, pages 748–759. Lect. Notes in Artificial Intelligence, 2143, Springer-Verlag, 2001.
2. M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *J. of Logic Programming*, 12:335–367, 1992.
3. J. Medina, M. Ojeda-Aciego, and P. Vojtáš. A completeness theorem for multi-adjoint logic programming. In *Proc. FUZZ-IEEE'01*. The 10th IEEE International Conference on Fuzzy Systems, IEEE Press, 2001. To appear.
4. J. Medina, M. Ojeda-Aciego, and P. Vojtáš. A multi-adjoint logic approach to abductive reasoning. In *Proc. 17th International Conference on Logic Programming, ICLP'01*. Lect. Notes in Artificial Intelligence 2273, Springer-Verlag, 2001.
5. J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Multi-adjoint logic programming with continuous semantics. In *Proc. Logic Programming and Non-Monotonic Reasoning, LPNMR'01*, pages 351–364. Lect. Notes in Artificial Intelligence, 2173, Springer-Verlag, 2001.
6. 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.
7. P. Vojtáš and L. Paulík. Soundness and completeness of non-classical extended SLD-resolution. In *Proc. Extensions of Logic Programming*, pages 289–301. Lect. Notes in Comp. Sci. 1050, Springer-Verlag, 1996.