Multi-Adjoint Logic Programming

J. Medina
Dept. Matemática Aplicada
Univ. de Málaga, Spain
jmedina@ctima.uma.es

M. Ojeda-Aciego
Dept. Matemática Aplicada
Univ. de Málaga, Spain
aciego@ctima.uma.es

Abstract

A synthesis of results of the recently introduced paradigm of multi-adjoint logic programming is presented. These results range from a proof theory together with some (quasi)completeness results to general termination results, and from the neural-like implementation of its fix-point semantics to the more general biresiduated multi-adjoint logic programming and its relationship with other approaches.

1 Introduction

Fuzzy logic is a powerful mathematical tool for dealing with modeling and control aspects of complex processes, as well as with uncertain, incomplete and/or inconsistent information. The main advantages of fuzzy logic systems are the capability to express nonlinear input/output relationships by a set of qualitative if-then rules, and to handle both numerical data and linguistic knowledge, which is extremely difficult to quantify by means of traditional mathematics.

A number of logic approaches have been introduced in the recent years motivated by the problem of reasoning in situations where information may be vague or uncertain, involving either fuzzy or annotated or probabilistic or similarity-based logic programming, e.g. [2, 9, 10, 11, 16, 18, 31, 34].

We present here a synthesis of results obtained from a lattice-valued logic programming paradigm called *multi-adjoint*, which permits the articulation of vague concepts and generalizes several approaches to the extension of logic programming techniques to the fuzzy case.

Multi-adjoint logic programming was introduced in [22] as a refinement of both [7, 34]. It allows for very general connectives in the body of the rules and, in addition, different types of implications to build the rules in the program. Such an approach is interesting for applications in which connectives depend on different users preferences; or in which knowledge is described by a many-valued logic program where connectives can be general aggregation operators (conjunctors, disjunctors, arithmetic mean, weighted sum, ...), even different aggregators for different users.

2 Multi-adjoint logic programs

The main concept on which multi-adjoint programs are built is that of *adjoint pair*.

Definition 1 *Let $\langle P, \preceq \rangle$ be a partially ordered set and $(\leftarrow, \&)$ a pair of binary operations in P such that:*

- *$\&$ is increasing in both arguments.*
- *$\leftarrow$ is increasing in the first argument (the consequent) and decreasing in the second argument (the antecedent).*
- *For any $x, y, z \in P$, we have that*

$$x \preceq (y \leftarrow z) \quad \textit{iff} \quad (x \,\&\, z) \preceq y$$

Then we say that $(\leftarrow, \&)$ forms an adjoint pair *in $\langle P, \preceq \rangle$.*

Extending the results in [7, 8, 34] 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. More formally,

Definition 2 (Multi-Adjoint Lattice) *Let $\langle L, \preceq \rangle$ be a lattice. A* multi-adjoint lattice $\mathcal{L}$ *is a tuple $(L, \preceq, \leftarrow_1, \&_1, \ldots, \leftarrow_n, \&_n)$ satisfying the following items:*

- *$\langle L, \preceq \rangle$ is bounded, i.e. it has bottom ($\bot$) and top ($\top$) elements;*
- *$(\leftarrow_i, \&_i)$ is an adjoint pair in $\langle L, \preceq \rangle$ for $i = 1, \ldots, n$;*
- *$\top \,\&_i\, \vartheta = \vartheta \,\&_i\, \top = \vartheta$ for all $\vartheta \in L$ for $i = 1, \ldots, n$.*

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

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 algebra.

In practice, we will usually have to assume some properties on the extra operators considered. These extra operators will be assumed to be either aggregators, or conjunctors or disjunctors, all of which are monotone functions (the latter, in addition, are required to generalize their Boolean counterparts). Note that the use of aggregators as weighted sums somehow covers the approach taken in [2] when considering the evidential support logic rules of combination.

The definition of multi-adjoint logic program is given, as usual, as a set of rules and facts. The particular syntax of these rules and facts is given below:

Definition 3 *A* multi-adjoint logic program *is a set of weighted rules $\langle A \leftarrow_i \mathcal{B}, \vartheta \rangle$ such that:*

1. *The* head *A is a propositional symbol.*
2. *The* body *formula $\mathcal{B}$ is a formula built from propositional symbols by the use of monotone operators.*
3. *The* weight *ϑ is an element of L.*

Facts *are rules with body $\top$.*

Definition 4

1. *An* interpretation *is a mapping I from the set of propositional symbols Π to the lattice $\langle L, \preceq \rangle$.*

   Note that each interpretation I can be uniquely extended to the whole set of formulas, and this extension is noted as $\hat{I}$.

2. *An interpretation I* satisfies *$\langle A \leftarrow_i \mathcal{B}, \vartheta \rangle$ if and only if $\vartheta \preceq \hat{I}\,(A \leftarrow_i \mathcal{B})$.*

3. *An interpretation I is a* model *of a multi-adjoint logic program $\mathbb{P}$ iff all weighted rules in $\mathbb{P}$ are satisfied by I.*

A procedural semantics for the general theory of multi-adjoint logic programming, together with two quasi-completeness theorems are presented in [23].

From the point of view of automation of the deduction process, this procedural semantics was not suitable for implementation. Instead, the fixpoint semantics has been chosen in [20] for implementation issues, this is the subject of the section below.

3 Neural implementation of multi-adjoint LP

A hybrid approach to handling uncertainty has been presented in [20], which is *expressed* in the rich language of multi-adjoint logic but is *implemented* by using ideas borrowed from the world of neural networks. This choice was motivated because of the have a massively parallel architecture-based dynamics of neural networks, inspired by the structure of human brain, adaptation capabilities, and fault tolerance.

Using neural networks in the context of logic programming is not a completely novel idea;

for instance, in [13] it is shown how fuzzy logic programs can be transformed into neural nets, where adaptations of uncertainties in the knowledge base increase the reliability of the program and are carried out automatically.

Regarding the approximation of the semantics of logic programs, in [14] the fixpoint of the $T_{\mathbb{P}}$ operator for a certain class of classical propositional logic programs (called acyclic logic programs) is constructed by using a 3-layered recurrent neural network, as a means of providing a massively parallel computational model for logic programming; this result is later extended in [15] to deal with the first order case.

The approach taken tries to join somehow the two approaches above, and it is interesting since our logic is much richer than classical or the usual versions of fuzzy logic in the literature.

The implementation makes use of a preprocessing stage, in which the initial rules are transformed into *homogeneous* form.

Definition 5 *A rule is said to be* homogeneous *if it has one of the following forms:*

- $\langle A \leftarrow_i \&_i(B_1, \dots B_n), \vartheta \rangle$
- $\langle A \leftarrow_i @(B_1, \dots, B_n), 1 \rangle$
- $\langle A \leftarrow_i B_1, \vartheta \rangle$

where A and B_i are propositional symbols.

A procedure to homogenize a given multi-adjoint program is presented and proved to fulfill the translation in linear time and space [21]. Then, transformation rules carry programs into neural networks, where truth-values of rules relate to output of neurons, truth-values of facts represent input, and network functions are determined by a set of general operators; the net outputs the values of propositional variables under the immediate consequences operator $T_{\mathbb{P}}$. By iterating the calculation it is possible to reproduce the successive iterated powers of $T_{\mathbb{P}}$ and, hence, to approximate the fixed point of $T_{\mathbb{P}}$ (the minimal model of $\mathbb{P}$) up to any level of precision.

It is remarkable that no learning capability is considered in the implementation of the minimal model semantics is given. Our approach being more complex than usual systems, concerning learning aspects it seems likely that some ideas from hybrid type networks should have to be taken into account, for instance reinforcement for fuzzy control like systems [3]. Learning is expected to be a key point when facing the general goal of this line of research, the development of a multi-adjoint approach to abductive logic programming.

4 Multi-Adjoint Languages and Similarity

A source of applications of the multi-adjoint languages is the study of similarity-based fuzzy unification. Several approaches have been proposed for similarity-based fuzzy unification we choose the way of including additional information about fuzzy similarities of different objects and using axioms of equality to transfer properties between these objects.

Based on the soundness and completeness of the declarative and procedural semantics of multi-adjoint logic programming, especially the fix-point semantics, it is possible to give a basis for a model of fuzzy unification. Our approach to fuzzy unification [25] is based on a theory of fuzzy logic programming with crisp unification constructed on the first-order multi-adjoint framework. On this computational model, a similarity-based unification approach is constructed by simply adding axioms of fuzzy similarities and using classical crisp unification which provides a semantic framework for logic programming with different notions of similarity.

It should be remarked there are different possible definitions of the concept of similarity all of them with their pros and cons, see [**?**] and its references. We will use the definition used in [29], where we will assume that $\langle L, \preceq \rangle$ denotes a lattice.

Definition 6 *A* similarity *on X is a mapping $s\colon X \times X \to L$ satisfying*

1. $s(x, x) = 1$ *(reflexivity)*

2. $s(x,y) = E(y,x)$ *(symmetry)*

3. $s(x,y) \wedge s(y,z) \preceq s(x,z)$ *(transitivity)*

for all $x, y, z \in X$.

Our approach considers similarities acting on elements of domains of attributes. The idea is based on the fact that part of our knowledge base, the multi-adjoint program $\mathbb{P}$, might consist of graded facts representing information about existent similarities on different domains which depend on the predicate they are used in, e.g. consider different spellings of a name in different languages (Salas, Salaš, Szałas, Szálás)

$$\begin{aligned} &\langle s(\text{Salas}, \text{Salaš}), 0.9\rangle \\ &\langle s(\text{Szałas}, \text{Szálás}), 0.9\rangle \\ &\langle s(\text{Salas}, \text{Szálás}), 0.8\rangle \quad \cdots \end{aligned}$$

The particular semantics of the multi-adjoint paradigm, enables one to easily implement a version of fuzzy unification by extending suitably the given program $\mathbb{P}$.

An extension of $\mathbb{P}$ is constructed by adding a parameterized theory (which introduces a number of similarities depending on the predicate and function symbols in $\mathbb{P}$), such as those below

$$\begin{aligned} \langle s(x,x), \top\rangle \\ \langle s(x,y) \leftarrow s(y,x), \top\rangle \\ \langle s(x,z) \leftarrow s(x,y) \,\&\, s(y,z), \top\rangle \end{aligned}$$

For all function symbol we also have

$$\begin{aligned} \langle s(f(x_1,\ldots,x_n), f(y_1,\ldots,y_n)) \leftarrow \\ s_1^f(x_1,y_1) \,\&\, \cdots \,\&\, s_n^f(x_n,y_n), \top\rangle \end{aligned}$$

Finally, given a predicate symbol, then the following rules are added

$$\begin{aligned} \langle P(y_1,\ldots,y_n) \leftarrow P(x_1,\ldots,x_n) \,\& \\ \& \, s_1^P(x_1,y_1) \,\&\, \cdots \,\&\, s_n^P(x_n,y_n), \top\rangle \end{aligned}$$

where & is some conjunction suitably describing the situation formalized by the program.

This way we get a multi-adjoint logic program $\mathbb{P}_E$ in which it is possible to get computed answers wrt $\mathbb{P}_E$ with similarity match in unification. Similarity-based computed answers are nothing but computed answers with crisp unification on a program extended by axioms of equality.

As an example of the flexibility of this formal approach, it is shown that the weak unification algorithm introduced in [29] can be emulated by our unification model. From a practical point of view, the proposed approach seems to be appropriate for some applications for information retrieval systems such as those studied in [19]. Moreover, in [17] different approaches were introduced to generate similarities to be used in flexible query answering systems, such as statistical generation of fuzzy similarities, or generation by some information retrieval techniques, or similarities arising from fuzzy conceptual lattices.

5 Termination results

An important issue regarding the computational behaviour of the proposed semantics is related to termination results. Specifically, when considering the fixpoint semantics (as it was implemented as stated in Section 3) it is interesting to obtain sufficient conditions under which the $T_{\mathbb{P}}$ operator attains its least fixpoint after finitely many iterations; this is the aim of [6], whose main contribution is the study of general properties guaranteeing termination.

The proposal uses a sorted version of a multi-adjoint language, where each sort identifies an underlying lattice of truth-values (weights) which must satisfy adjoint conditions. This seems very appropriate for performing and representing several reasoning tasks with imprecise and incomplete information, and is based on probabilistic deductive databases proposed in [18]. The semantics of sorted multi-adjoint logic program is characterised, as usual, by the post-fixpoints of the immediate consequence operator $T_{\mathbb{P}}$.

The major contributions of [6] are the termination results for several classes of sorted multi-adjoint logic programs, extending or complementing existing results in the litera-

ture [9, 10, 16, 18, 28]. In particular, the case of programs obtained by arbitrary composition of operators obeying the boundary condition $\vartheta \otimes 1 = 1 \otimes \vartheta \leq \vartheta$ over the unit interval are shown to be terminating.

A practical application of Multi-Adjoint Σ-Algebras can be found in the probabilistic deductive databases framework of Lakshmanan and Sadri [18] where our sorts correspond to disjunctive modes and the adjoint operators to different conjunctive modes for combining probabilistic knowledge. The proposed framework is richer since we do not restrain ourselves to a single and particular carrier set and allow more operators.

6 Biresiduation, gaggles, implication triples

Many different "and" and "or" operations have been proposed for use in fuzzy logic. Several papers discuss the optimal choice of these operations for fuzzy control, when the main criterion is to get the most stable control. In reasoning applications, however, it is more appropriate to select operations which are the best in reflecting human reasoning, i.e., operations which are "the most logical". The need of biresiduated pairs is justified in [24] by presenting analytical evidence of reasonable non-commutative (moreover, non-associative) conjunctors, which lead to the consideration of two residuated implications.

Building on the fact that conjunctors in multi-adjoint logic programs need not be either commutative or associative, a further generalization of the multi-adjoint framework was presented, introducing *biresiduated* multi-adjoint logic programming and its fixpoint semantics.

To the best of our knowledge, there is not much work done using biresidua in the sense considered in [24]. Recently, in [32] a study of the representability of biresiduated algebras was presented. This type of algebras were introduced in a purely algebraic context, being studied for instance in [4], and, in the framework of substructural logics, in [12] under the name of *(partial) gaggles*. On a different basis, in [26] an axiom system is developed based on a biresiduation construction (called *implication triple*) and the completeness and soundness theorems are proved; in [30], a structure of biresiduated algebra is defined together with a corresponding logical system regarding the use of fuzzy sets in the representation of multicriteria decision problems.

The biresiduated multi-adjoint framework is based on the definitions of biresiduated triple and biresiduated multi-adjoint lattice.

Definition 7 *A* biresiduated triple *on L is an ordered triple $(\&, \swarrow, \nwarrow)$ in which $\&$ is a conjunction on L and $\swarrow$ and $\nwarrow$ are implications on L, mutually related by the following adjointness condition*

$$\beta \leq \gamma \nwarrow \alpha \text{ iff } \alpha \,\&\, \beta \leq \gamma \text{ iff } \alpha \leq \gamma \swarrow \beta$$

Definition 8 *Let $\langle L, \preceq \rangle$ be a lattice. A* biresiduated multi-adjoint lattice *$\mathcal{L}$ is a tuple $(L, \preceq, \swarrow 1, \nwarrow_1, \&_1, \ldots, \swarrow n, \nwarrow_n, \&_n)$ satisfying the following items:*

1. *$\langle L, \preceq \rangle$ is bounded, i.e. it has bottom and top elements;*
2. *$(\&_i, \swarrow i, \nwarrow_i)$ are biresiduated triples, for all $i = 1, \ldots, n$.*

The adjointness conditions of a biresiduated triple, make this algebraic structure a flexible and suitable tool for being used in a logical context, for they can be interpreted as a two possible multiple-valued *modus ponens*-like rules. From a categorical point of view, these conditions arise when considering the conjunctor as a bifunctor, and applying the adjointness either in its second or first argument, respectively.

The structure of biresiduated multi-adjoint lattice is the basis of the definition of biresiduated multi-adjoint logic program. Some practical applications are envisaged for the obtained results, such as the development of a generalized multiple-valued resolution. In a generalized context it is not possible to deal with Horn clauses and refutation, 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 would allow the development of a sound and complete graded resolution.

7 Conclusions and future work

Multi-adjoint logic programs are capable of capturing and combining several reasoning paradigms dealing with imprecision and uncertainty. A synthesis of recent results obtained for the framework of multi-adjoint logic has been presented. There are still a number of issues which are worth to be investigated.

The embedding of other proposals in the literature into our framework has to be explored in subsequent work.

Regarding the neural implementation, a theoretical analysis of the convergence rate of the net to the minimal model have to be developed, at least for some particular classes of multi-adjoint programs. At a different level, the consideration of the learning capabilities of the net are providing promising results for diagnosis/abduction problems, which encourages further research in this area.

The study of termination of the semantics has to be extended to considering tabling proof procedures for first-order sorted multi-adjoint logic programs, relying mainly in the above cited results.

A thorough study of biresiduated triples (under the name of *implication triples*) has been recently developed in [27]. The interaction among the theoretical study of implication triples in the setting of biresiduated multi-adjoint logic programs has to be studied.

References

[1] A.S. d'Avila Garcez, K. Broda and D.M. Gabbay. *Neural-Symbolic Learning Systems: Foundations and Applications*. Perspectives in Neural Computing, Springer-Verlag, 2002.

[2] J.F. Baldwin, T.P. Martin, and B.W. Pilsworth. *Fril - Fuzzy and Evidential Reasoning in Artificial Intelligence*. Research Studies Press Ltd, 1995.

[3] H.R. Berenji and P. Khedkar. Learning and tuning fuzzy logic controllers through reinforcements. *IEEE Tr. on Neural Networks*, 3(5):724–740, 1992.

[4] W. J. Blok and S. B. la Falce. Komori identities in algebraic logic. *Reports on Mathematical Logic*, 34:79–106, 2001.

[5] C.V. Damásio and M. Ojeda-Aciego. On termination of a tabulation procedure for residuated logic programming. 6th Intl Workshop on Termination, pp. 40-43, 2003

[6] C.V. Damásio, J. Medina, and M. Ojeda-Aciego. Termination results for sorted multi-adjoint logic programming. Submitted to IPMU'04, 2004

[7] C.V. Damásio and L.M. Pereira. Monotonic and residuated logic programs. *Lect. Notes in Artificial Intelligence* 2143, pp. 748–759, 2001.

[8] C.V. Damásio and L.M. Pereira. Hybrid probabilistic logic programs as residuated logic programs. *Studia Logica*, 72(1):113–138, 2002.

[9] M. Dekhtyar, A. Dekhtyar and V.S. Subrahmanian. Hybrid Probabilistic Programs: Algorithms and Complexity. Proc. of Uncertainty in AI'99 conference, 1999

[10] A. Dekhtyar and V.S. Subrahmanian. Hybrid Probabilistic Programs, *Journal of Logic Programming* 43(3):187–250, 2000

[11] D. Dubois, J. Lang and H. Prade. Towards Possibilistic Logic Programming. Proc. of International Conference on Logic Programming, pp. 581–598, MIT Press, 1991

[12] J. M. Dunn. Partial gaggles applied to logics with restricted structural rules. In K. Došen and P. Schroeder-Heister, editors, *Substructural Logics*, Studies in Logic and Computation, pages 63–108. Oxford University Press, 1993.

[13] P. Eklund and F. Klawonn. Neural fuzzy logic programming. *IEEE Tr. on Neural Networks*, 3(5):815–818, 1992.

[14] S. Hölldobler and Y. Kalinke. Towards a new massively parallel computational model for logic programming. In *ECAI'94 workshop on Combining Symbolic and Connectioninst Processing*, pages 68–77, 1994.

[15] S. Hölldobler, Y. Kalinke, and H.-P. Störr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence*, 11(1):45–58, 1999.

[16] M. Kifer and V.S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *J. of Logic Programming* 12(4):335–367, 1992

[17] S. Krajči, R. Lencses, J. Medina, M. Ojeda-Aciego, and P. Vojtáš. A similarity-based unification model for flexible querying. *Lect. Notes in Artificial Intelligence* 2522:263–273, 2002.

[18] L. Lakhsmanan and F. Sadri. On a theory of probabilistic deductive databases. *Theory and Practice of Logic Programming* 1(1):5–42, 2001

[19] R. Lencses. Algoritmy v oblasti získavania informácii. In *Proc. Information Technologies — Applications and Theory*, pages 53–64. Faculty of Science. UPJS, Košice, Slovakia, 2001.

[20] J. Medina, E. Mérida-Casermeiro, and M. Ojeda-Aciego. A Neural Implementation of Multi-Adjoint Logic Programming. To appear in *Journal of Applied Logic*, 2004.

[21] J. Medina, and M. Ojeda-Aciego. Homogenizing multi-adjoint logic programs. Intl Conference on Fuzzy Logic and Technology (EUSFLAT'03), pp. 640–644, 2003.

[22] J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Multi-adjoint logic programming with continuous semantics. *Lect. Notes in Artificial Intelligence* 2173, pp. 351–364, 2001.

[23] J. Medina, M. Ojeda-Aciego, and P. Vojtáš. A procedural semantics for multi-adjoint logic programming. *Lect. Notes in Artificial Intelligence* 2258, pp. 290–297, 2001.

[24] J. Medina, M. Ojeda-Aciego, A. Valverde, and P. Vojtáš. Towards biresiduated multi-adjoint logic programming. Post-conference proceedings of the Spanish Conference on AI, to appear in *Lect. Notes in Artificial Intelligence*, 2004.

[25] J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Similarity-based unification: a multi-adjoint approach. To appear in *Fuzzy Sets and Systems*, 2004.

[26] N.N. Morsi. Propositional calculus under adjointness. *Fuzzy Sets and Systems*, 132:91–106, 2002.

[27] N.N. Morsi, and E.M. Roshdy. Implication triples. To appear in *Fuzzy Sets and Systems*, 2004.

[28] L. Paulík. Best possible answer is computable for fuzzy SLD-resolution. *Lecture Notes on Logic* 6, pp. 257–266, 1996.

[29] M.I. Sessa. Approximate reasoning by similarity-based SLD resolution. *Theoretical Computer Science*, 275(1–2):389–426, 2002.

[30] M. Sularia. A logical system for multicriteria decision analysis. *Studies in Informatics and Control*, 7(3), 1998.

[31] M.H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 4(1):37–53, 1986.

[32] C. J. van Alten. Representable biresiduated lattices. *Journal of Algebra*, 247:672–691, 2002.

[33] A. van Gelder. Negation as failure using tight derivations for general logic programs. In J. Minker (ed.) *Foundations of deductive databases and logic programming*, pp. 149–176, Morgan Kaufmann Publishers Inc., 1988

[34] P. Vojtáš. Fuzzy logic programming. *Fuzzy Sets and Systems*, 124(3):361–370, 2001.