Revisiting reductants in the multi-adjoint logic programming framework ⋆

P. Julián-Iranzo[1], J. Medina[2], and M. Ojeda-Aciego[3]

[1] Universidad de Castilla-La Mancha. Dept. Information Technologies and Systems.
[2] Universidad de Cádiz. Dept. de Matemáticas
[3] Universidad de Málaga. Dept. de Matemática Aplicada

Abstract. Reductants are a special kind of fuzzy rules which constitute an essential theoretical tool for proving correctness properties. A multi-adjoint logic program, when interpreted on an arbitrary lattice, has to include all its reductants in order to preserve the approximate completeness property.
In this work, after revisiting the different notions of reductant arisen in the framework of multi-adjoint logic programming and akin frameworks, we introduce a new, more adequate, notion of reductant in the context of multi-adjoint logic programs. We study some of its properties and give an efficient algorithm for computing all the reductants associated with a multi-adjoint logic program.

Keywords: Fuzzy Logic Programming, Multi-adjoint Logic Programming, Reductants.

1 Introduction

Fuzzy extensions of the logic programming paradigm have been investigated since the late eighties and the decade of the nineties [1, 4, 7, 11, 22]; later, some general frameworks were introduced and their interrelationships were studied [2, 5, 12, 14, 19, 21]; currently, one can still find papers on the subject of fuzzy logic programming, some of them even from the perspective of category theory, which address important issues in this topic [3, 13, 17]. This work focuses on multi-adjoint logic programming [15] and, specifically, on the most adequate notion of reductant for a logic program.

Multi-adjoint logic programming is a flexible framework combining fuzzy logic and logic programming. Roughly speaking, a multi-adjoint logic program can be seen as a set of implicational rules annotated by a truth degree (a value of a complete lattice). One of the main features of the multi-adjoint framework is its flexibility, in that rules need not be written with a common implications, and the most suitable one can be used instead; another important feature is that it works even when the conjunctors used in the body of the rules are neither commutative or associative.

⋆ Partially supported by the Spanish MICINN projects TIN2012-39353-C04-01, TIN2012-39353-C04-04, TIN2011-25846 and the Spanish Ministry of Economy and Competition under grant TIN2013-45732-C4-2-P.

Reductancts were first introduced in the context of generalized annotated logic programming [11] in order to deal with problems related to incompleteness. The multi-adjoint logic programming paradigm has to deal with a similar problem of incompleteness that may arise when programs are interpreted in a non-linear lattice. Specifically, it might be not possible to compute the greatest correct answer (for a given goal and program) due to the existence of incomparable elements in $(L, \preceq)$, see [15]. As a result, multi-adjoint programs need to incorporate a special kind of rules, called *reductants*, in order to preserve the (approximate) completeness property, and this introduces severe penalties in the implementation of efficient multi-adjoint logic programming systems, since not only the size of programs increases but also their execution time. Moreover, the original definitions of reductants often produce infinitely many reductants for some programs. Therefore, if we want to develop complete and efficient implementation systems for the multi-adjoint logic framework, it is essential to define more accurate notions of reductants and methods for optimizing their computation.

In this work, after revisiting different notions of reductant proposed for multi-adjoint programs, we define a new, more adequate, notion of reductant. We study some of its formal properties and give an efficient algorithm for computing all the reductants associated with a multi-adjoint logic program.

2 Syntax and Semantics of Multi-adjoint Logic Programs

We will work with a first order language, $\mathcal{L}$, containing variables, function symbols, predicate symbols, constants, the classical quantifiers ($\forall$ and $\exists$), and several (arbitrary) connectives in order to increase language expressiveness.

In our fuzzy setting, we assume a number of implication connectives ($\leftarrow_i$) together with other connectives, so-called "aggregators" (usually denoted $@_j$), used to build the bodies of the rules. The general definition of aggregation operators subsumes conjunctive operators (denoted by $\&_k$), disjunctive operators ($\vee_l$), and average and hybrid operators. The truth function for an n-ary aggregation operator[1] $@ : L^n \to L$ is required to be monotone and fulfill $@(\top, \dots, \top) = \top$, $@(\bot, \dots, \bot) = \bot$. The underlying set of truth-values is assumed to be a complete lattice L together with a collection of adjoint pairs intended to produce the evaluation of *modus ponens* [6].

A *rule* is a formula $H \leftarrow_i \mathcal{B}$, where H is an atomic formula (usually called the *head*) and $\mathcal{B}$ (which is called the *body*) is a formula built from atomic formulas $B_1, \dots, B_n$, $n \geq 0$, truth values of L and aggregation operators. Rules whose body is $\top$ are called *facts* (usually, we will represent a fact as a rule with an empty body). A *goal* is a body submitted as a query to the system. Variables in a rule are assumed to be universally quantified. Roughly speaking, a multi-adjoint logic program is a set of pairs $\langle \mathcal{R}; \alpha \rangle$, where $\mathcal{R}$ is a rule and α is a *weight*, usually assigned by an expert.

[1] Note that, as no confusion arises, we use the same notation for a formal function symbol and its semantic meaning.

Formulas are interpreted on a multi-adjoint lattice. In this framework, it is sufficient to consider Herbrand interpretations, in order to define a declarative semantics. See [15] for a formal characterization of a *fuzzy interpretation*, $\mathcal{I}$, as a mapping from the Herbrand base, $B_{\mathcal{L}}$, into the multi-adjoint lattice of truth values L and a notion of evaluation and satisfiability of formulas.

The procedural semantics can be formalized as an operational phase followed by an interpretive one. The operational phase uses a residuum-based generalization of *modus ponens* [6] that, given an atomic goal A and a program rule $\langle H \leftarrow_i \mathcal{B}; v\rangle$, if there is a most general unifier substitution $\theta = mgu(\{A = H\})$ the atom A is substituted by the expression $(v \&_i \mathcal{B})\theta$. In the following, we write $\mathcal{C}[A]$ to denote a formula where A is a sub-expression (usually an atom) occuring in the—possibly empty—context $\mathcal{C}[]$. Moreover, $\mathcal{C}[A/H]$ means the replacement of A by H in context $\mathcal{C}[]$. Also we use $\mathcal{V}ar(s)$ for referring to the set of variables occurring in the syntactic object s, whereas $\theta[\mathcal{V}ar(s)]$ denotes the substitution obtained from θ by restricting its domain, $Dom(\theta)$, to $\mathcal{V}ar(s)$.

Definition 1 (Admissible Steps). *Let $\mathcal{Q}$ be a goal and let σ be a substitution. The pair $\langle \mathcal{Q}; \sigma\rangle$ is a* state *and we denote by $\mathcal{E}$ the set of states. Given a program $\mathcal{P}$, an* admissible computation *is formalized as a state transition system, whose transition relation $\leadsto_{AS} \subseteq (\mathcal{E} \times \mathcal{E})$ is the smallest relation satisfying the following* admissible rules *(where we consider that A is the selected atom in $\mathcal{Q}$):*

1) $\langle \mathcal{Q}[A]; \sigma\rangle \leadsto_{AS} \langle (\mathcal{Q}[A/v\&_i\mathcal{B}])\theta; \sigma\theta\rangle$ *if* $\theta = mgu(\{H = A\})$, $\langle H \leftarrow_i \mathcal{B}; v\rangle$ *in* $\mathcal{P}$.
2) $\langle \mathcal{Q}[A]; \sigma\rangle \leadsto_{AS} \langle (\mathcal{Q}[A/\bot]); \sigma\rangle$ *if there is no rule in $\mathcal{P}$ whose head unifies A.*

Formulas involved in admissible computation steps are renamed apart before being used. The symbols $\leadsto^{+}_{AS}$ and $\leadsto^{*}_{AS}$ denote, respectively, the transitive closure and the reflexive, transitive closure of $\leadsto_{AS}$.

Definition 2. *Let $\mathcal{P}$ be a program and let $\mathcal{Q}$ be a goal. An* admissible derivation *is a sequence $\langle \mathcal{Q}; id\rangle \leadsto^{*}_{AS} \langle \mathcal{Q}'; \theta\rangle$. When $\mathcal{Q}'$ is a formula not containing atoms, the pair $\langle \mathcal{Q}'; \sigma\rangle$, where $\sigma = \theta[\mathcal{V}ar(\mathcal{Q})]$, is called an* admissible computed answer *(a.c.a.) for that derivation.*

If we exploit all atoms of a goal, by applying admissible steps as much as needed during the operational phase, then it becomes a formula with no atoms which can be then directly interpreted in the multi-adjoint lattice L.

Definition 3 (Interpretive Step). *Let $\mathcal{P}$ be a program, $\mathcal{Q}$ a goal and σ a substitution. We formalize the notion of* interpretive computation *as a state transition system, whose transition relation $\leadsto_{IS} \subseteq (\mathcal{E} \times \mathcal{E})$ is the smallest one satisfying: $\langle Q[@(r_1, \dots, r_n)]; \sigma\rangle \leadsto_{IS} \langle Q[@(r_1, \dots, r_n)/v]; \sigma\rangle$, where v is the truth value obtained after evaluating $@(r_1, \dots, r_n)$ in the lattice $\langle L, \preceq\rangle$ associated with $\mathcal{P}$.*

Definition 4. *Let $\mathcal{P}$ be a program and $\langle Q; \sigma\rangle$ an a.c.a., that is, $\mathcal{Q}$ is a goal not containing atoms. An* interpretive derivation *is a sequence $\langle Q; \sigma\rangle \leadsto^{*}_{IS} \langle Q'; \sigma\rangle$. When $Q' = r \in L$, $\langle L, \preceq\rangle$ being the lattice associated with $\mathcal{P}$, the state $\langle r; \sigma\rangle$ is called a* fuzzy computed answer *(f.c.a.) for that derivation.*

We denote by $\rightsquigarrow_{IS}^{+}$ and $\rightsquigarrow_{IS}^{*}$ the transitive closure and the reflexive, transitive closure of $\rightsquigarrow_{IS}$, respectively. Also note that, sometimes, when it is not important to pay attention on the substitution component of a f.c.a. $\langle r; \theta\rangle$ (maybe, because $\theta = id$) we shall refer to the value component r as the "f.c.a.".

3 Different notions of reductant

In this section we survey the different notions of reductants raised over the last years in the field of fuzzy logic programming, describing some of their features which are important for the present work.

The original notion of reductant appeared in the framework of generalized annotated logic programming [11] was initially adapted to the multi-adjoint logic programming framework in the following terms [15]:

Definition 5 (Reductant). *Let $\mathcal{P}$ be a program, A a ground atom, and the (non empty) set of rules $\{\langle C_i \leftarrow_i \mathcal{B}_i; v_i\rangle \mid 1 \leq i \leq n\}$ in $\mathcal{P}$ whose head matches with A (i.e., for each C_i there exists a θ_i such that $A = C_i\theta_i$). A reductant for A in $\mathcal{P}$ is a rule $\langle A \leftarrow @(\mathcal{B}_1, \ldots, \mathcal{B}_n)\theta; \top\rangle$ where $\theta = \theta_1 \cdots \theta_n$, the connective $\leftarrow$ is any implication with an adjoint conjunctor, and the truth function for the intended aggregator $@$ is defined as $@(b_1, \ldots, b_n) = sup\{v_1 \&_1 b_1, \ldots, v_n \&_n b_n\}$.*

This notion was introduced as a valuable theoretical tool for proving the (approximate) completeness property of the multi-adjoint logic programming framework. It is worth to note that, contrariwise to the original definition of a reductant in [11], which is uniquely linked with a program, this one is linked to a ground atom and a program.

In order to preserve the approximate completeness property, it is necessary to construct the "completion" of a program, extending it with all their reductants. So, if one has to compute all the reductants associated with a program, all the atoms of the Herbrand base of that program, which might be infinite, should be taken into account. Hence, although this notion of reductant is theoretically valuable may easily turn impractical because of its potential non-termination. Therefore, it was soon clear that if we wanted to implement complete systems we needed a new notion of reductant leading to finite completions and producing reductants able to be executed more efficiently.

An alternative version of reductant, named PE-reductant, has been proposed in the literature [10] using partial evaluation techniques [8]. Despite PE-reductants may improve the efficiency of multi-adjoint computations they do not overcome the need to compute an infinite number of reductants, since they continue linked to the Herbrand base of ground atoms.

As a step further in the path of trying to avoid the proliferation of an infinite number of reductants, in [16, 17], a new notion named G-reductant was introduced. The aim was that a single generalized reductant was required to cover all the (possibly infinite) calls to atoms headed by a specific predicate symbol defined in a program.

Definition 6 (G-Reductant). *Given a program $\mathcal{P}$ and a definite predicate p in $\mathcal{P}$, a G-reductant for the predicate p in $\mathcal{P}$ is a rule*

$$\langle p(X_1, \dots, X_m) \leftarrow @(\hat{\theta}_1 \& \mathcal{B}_1, \dots, \hat{\theta}_n \& \mathcal{B}_n); \top\rangle$$

where

– $\{\langle C_i \leftarrow_i \mathcal{B}_i; v_i\rangle \mid 1 \leq i \leq n\}$ *is the non-empty set of rules such that every* C_i *is an instance of* $p(X_1, \dots, X_m)$ *via the substitution* $\theta_i = \{X_1/t_{i1}, \dots, X_m/t_{im}\}$.
– $\hat{\theta}_i \equiv (X_1 \approx t_{i1} \& \cdots \& X_m \approx t_{im})$ *with* $\approx$ *being a unification operator defined by the rule* $R_{\approx} \equiv \langle X \approx X; \top\rangle$*, which is considered to be included in every multi-adjoint program.*
– *the connective* $\leftarrow$ *is any implication with an adjoint conjunction* $\&$*, and the truth function for the intended aggregator* $@$ *is the same as in Definition 5.*

Observe that, although only finitely many G-reductants are generated for a given program (just one for each definite predicate in the program), due to the fact that they are built in a non-evaluated form, computing with this kind of reductants becomes inefficient. By this reason, in [18], unfolding-based techniques were applied for simplifying general reductants: the idea was to perform computational steps on the body of G-reductants, at transformation time, in order to improve their efficiency at execution time.

Despite the accomplishments obtained by these transformation techniques, the overall process is little intuitive and, what is worst, it does not guarantee the approximate completeness of a multi-adjoint logic programming framework.

4 A new notion of reductant: sets of critical rules

In this section we propose a new notion of reductant, once again in the line of [11], aiming at solving the aforementioned problems inherent to the other notions of reductant. We seek a new notion of reductant such that:

1. is not attached to a certain kind of goals for its computation,
2. can be computed efficiently, and
3. there is no need to consider infinitely many of them.

To begin with, we will informally discuss the underlying idea, and then proceed with the formal definition. Firstly, note that the need of using reductants arises when for a program $\mathcal{P}$ and an atom A (with or without variables) launched as a goal, there exist different derivations leading to fuzzy computed answers with the same computed substitution but leading to incomparable truth-values: $\langle v_1; \theta\rangle, \dots, \langle v_n; \theta\rangle$. In this case, $\langle \sup\{v_1, \dots, v_n\}; \theta\rangle$ can be proven to be a fuzzy correct answer which not computed by the operational mechanism.

To better understand this problem and to identify its source, we must investigate whether there exists some relationship between the program rules that take part in the derivations that cause the problem. To simplify the discussion, we analyze just the plain situation of a program containing two fact rules $\mathcal{R}_1 \equiv \langle H_1, a\rangle$,

$\mathcal{R}_2 \equiv \langle H_2, b \rangle$, where a and b are incomparable elements of a partially ordered multi-adjoint lattice, and a non-ground atom A launched as a goal to be solved. In order to reproduce the problem it should be possible to perform admissible steps both with $\mathcal{R}_1$ and $\mathcal{R}_2$ computing the fuzzy computer answers $\langle a; \theta \rangle$ and $\langle b; \theta \rangle$ respectively. But this is only possible if the heads of $\mathcal{R}_1$ and $\mathcal{R}_2$ unify with A with the same substitution θ, i.e. if $A\theta = H_1\theta$ and $A\theta = H_2\theta$. Therefore, since $H_1\theta = H_2\theta$, the heads of $\mathcal{R}_1$ and $\mathcal{R}_2$ unify. This is an interesting observation that gives a criterion to decide when a set of rules may cause the problem. In general the problem may occur when there exist sets of rules in a program whose heads unify. We shall say that such sets are "sets of critical rules".

Definition 7 (Critical Rules). *Let $\mathcal{P}$ be a program, and $\mathcal{R}_1 \equiv \langle H_1 \leftarrow \mathcal{B}_1, v_1 \rangle$, and $\mathcal{R}_2 \equiv \langle H_2 \leftarrow \mathcal{B}_2, v_1 \rangle$ two rules in $\mathcal{P}$ that are renamed apart. The rules $\mathcal{R}_1$ and $\mathcal{R}_2$ are said to be* critical *iff H_1 and H_2 unify, that is, there exists a substitution $\theta = mgu\{H_1, H_2\} \not\equiv fail$.*

A set of rules in $\mathcal{P}$, is a set of critical rules *iff the set of their heads unify.*

Note that a set of critical rules is composed by a subset of rules defining a certain predicate p in $\mathcal{P}$.

Example 1. Let $\mathcal{P} = \{\mathcal{R}_1 : \langle p(a, g(Z)) \leftarrow_1; v_1 \rangle, \mathcal{R}_2 : \langle p(Y, g(Y)) \leftarrow_2; v_2 \rangle\}$ be a program. The rules $\mathcal{R}_1$ and $\mathcal{R}_2$ are critical rules, since $mgu\{p(a, g(Z)), p(Y, g(Y))\} = \{Y/a, Z/a\} \not\equiv fail$. $\square$

Now we can introduce the new notion of reductant.

Definition 8 (Critical Reductant). *Let $\mathcal{P}$ be a program and $\{\langle H_i \leftarrow_i \mathcal{B}_i; v_i \rangle \mid 1 \leq i \leq n\}$ a set of critical rules in $\mathcal{P}$ with $\theta = mgu\{H_1, \ldots, H_n\}$. Then, the rule $\langle H_1\theta \leftarrow @_{sup}(v_1 \&_1 \mathcal{B}_1, \ldots, v_n \&_n \mathcal{B}_n)\theta; \top \rangle$ is a* critical reductant *of $\mathcal{P}$, where the connective $\leftarrow$ is any implication with an adjoint conjunctor, and the truth function for the aggregator $@_{sup}$ is the supremum operator.*

It is worth to recall that we are assuming an extended language where truth degrees and adjoint conjunctions are allowed in the body of program rules.

Observe that for programs with finitely many rules there always exist finitely many critical reductants. If $\mathcal{P}_p$ is the set of rules defining a predicate p, the elements in the powerset of $\mathcal{P}_p$ (excluding the empty set and the singletons) are the candidates to generate critical reductants, but only those which form sets of critical rules truly generate them. The sets of critical rules ordered by inclusion form a partially ordered set. The critical reductants obtained from the maximal elements of that set of sets will be called *maximal reductants*.

Example 2. For the program $\mathcal{P}$ of Example 1 there exists just one reductant, which is obtained from the maximal set of critical rules $\{\mathcal{R}_1, \mathcal{R}_2\}$:

$$\langle p(a, g(a)) \leftarrow sup\{v_1, v_2\}; \top \rangle.$$

Therefore, this is a maximal reductant.

5 Formal properties of critical reductants

In this section we establish some important properties of critical reductants which are substantive for the correctness of the multi-adjoint logic programming framework. The first result is a technical lemma, which will be used later.

Lemma 1. *Let $\mathcal{A}$ be a formula, $\mathcal{I}$ an interpretation and θ a substitution. Then $\mathcal{I}(\mathcal{A}) \leq \mathcal{I}(\mathcal{A}\theta)$.*

Proof. Immediate, since $\mathcal{I}(\mathcal{A}) = \inf\{\mathcal{I}(\mathcal{A}\xi) \mid \mathcal{A}\xi \text{ is a ground instantiation of } \mathcal{A}\}$, by definition of interpretation (see [15]), and

$$\begin{aligned}&\{\mathcal{I}(\mathcal{A}\theta\xi') \mid \mathcal{A}\theta\xi' \text{ is a ground instantiation of } \mathcal{A}\} \subseteq \\ &\qquad \subseteq \{\mathcal{I}(\mathcal{A}\xi) \mid \mathcal{A}\xi \text{ is a ground instantiation of } \mathcal{A}\}\end{aligned}$$ □

Proposition 1. *If $\mathcal{R}$ is a critical reductant of a multi-adjoint program $\mathcal{P}$, then every interpretation $\mathcal{I}$ which is a model of $\mathcal{P}$ is also a model of the critical reductant $\mathcal{R}$, that is, $\mathcal{P} \models \mathcal{R}$.*

Proof. Firstly note that, by definition of critical reductant (Defn. 8) there must exist a set $S = \{\langle H_i \leftarrow_i \mathcal{B}_i; v_i\rangle \mid 1 \leq i \leq n\}$ of critical rules in $\mathcal{P}$, whose heads unify and $\theta = mgu\{H_1, \ldots, H_n\}$, such that

$$\mathcal{R} \equiv \langle H_1\theta \leftarrow @_{sup}(v_1 \&_1 \mathcal{B}_1, \ldots, v_n \&_n \mathcal{B}_n)\theta; \top\rangle.$$

On the other hand, if an interpretation $\mathcal{I}$ is a model of $\mathcal{P}$, it is a model of all rules in $\mathcal{P}$, and specifically of the critical rules in S. Therefore, by definition of model (see [15]), we have $\mathcal{I}(H_i \leftarrow_i \mathcal{B}_i) \geq v_i$ and, by Lemma 1, $\mathcal{I}(H_i\theta \leftarrow_i \mathcal{B}_i\theta) \geq v_i$. Now, applying adjointness, it follows that $\mathcal{I}(v_i \&_i \mathcal{B}_i\theta) \leq \mathcal{I}(H_i\theta) = \mathcal{I}(H_1\theta)$.

Finally, note that, by the previous lemma, $\mathcal{I}(H_1\theta)$ is a upper bound of the set $\{v_1 \&_1 \mathcal{I}(\mathcal{B}_1\theta), \ldots, v_n \&_n \mathcal{I}(\mathcal{B}_n\theta)\}$ so

$$@_{sup}(v_1 \&_1 \mathcal{I}(\mathcal{B}_1\theta), \ldots, v_n \&_n \mathcal{I}(\mathcal{B}_n\theta)) \leq \mathcal{I}(H_1\theta).$$

therefore, as we are working with residuated implications

$$\mathcal{I}(H_1\theta \leftarrow @_{sup}(v_1 \&_1 \mathcal{B}_1, \ldots, v_n \&_n \mathcal{B}_n)\theta) = \top$$

and we can affirm that $\mathcal{I}$ is a model of the reductant $\mathcal{R}$. □

The converse result is not true in general; in fact, the natural requirement for it to be true is very restrictive, as we will show later.

Given a program $\mathcal{P}$, the set of the critical reductants of $\mathcal{P}$ will be denoted as $\mathcal{P}_R$, and the following result shows a procedure in order to obtain a model from a reductant program $\mathcal{P}_R$.

Proposition 2. *Given a multi-adjoint program $\mathcal{P}$ and a model $\mathcal{I}$ of $\mathcal{P}_R$, if $\mathcal{I}(A) = \inf\{\mathcal{I}(A\theta) \mid \langle A\theta \leftarrow @_{sup}(v_1 \&_1 \mathcal{B}_1, \ldots, v_n \&_n \mathcal{B}_n)\theta; \top\rangle \text{ is a critical reductant}\}$ then $\mathcal{I}$ is a model of $\mathcal{P}$.*

Proof. Given a rule $\langle A{\leftarrow_i}\ \mathcal{B}; v\rangle$ in $\mathcal{P}$ and a model $\mathcal{I}$ of $\mathcal{P}_R$, we need to prove that $v \&_i \mathcal{I}(\mathcal{B}) \leq \mathcal{I}(A)$. If no critical reductant with head $A\theta$ exist for a substitution θ, then, by hypothesis, $\mathcal{I}(A) = \top$ and the inequality is trivially satisfied.

Otherwise, since $\mathcal{I}$ is a model of $\mathcal{P}_R$, for all critical reductant

$$\langle A\theta{\leftarrow}\ @_{sup}(v_1\&_1\mathcal{B}_1, \dots, v_n\&_n\mathcal{B}_n)\theta; \top\rangle$$

the inequality $\mathcal{I}(@_{sup}(v_1\&_1\mathcal{B}_1, \dots, v_n\&_n\mathcal{B}_n)\theta) \leq \mathcal{I}(A\theta)$ holds and so, by Lemma 1, we have:

$$\begin{aligned} v\&_i\mathcal{I}(\mathcal{B}) &\leq \mathcal{I}(@_{sup}(v_1\&_1\mathcal{B}_1, \dots, v_n\&_n\mathcal{B}_n)) \\ &\leq \mathcal{I}(@_{sup}(v_1\&_1\mathcal{B}_1, \dots, v_n\&_n\mathcal{B}_n)\theta) \\ &\leq \mathcal{I}(A\theta) \end{aligned}$$

for each critical reductant. Consequently, by hypothesis, the result is obtained.

$$\begin{aligned} v\&_i\mathcal{I}(\mathcal{B}) \leq \inf\{&\mathcal{I}(A\theta) \mid \\ &\langle A\theta{\leftarrow}\ @_{sup}(v_1\&_1\mathcal{B}_1, \dots, v_n\&_n\mathcal{B}_n)\theta; \top\rangle \text{ is a critical reductant}\} \\ = \mathcal{I}(A)& \end{aligned}$$

□

The following proposition relates the notion of critical reductant and the G-reductant developed in [16, 17]. We claim the equivalence between maximal critical reductants of a multi-adjoint program $\mathcal{P}$ and the G-reductants of $\mathcal{P}$, after a sequence of unfolding steps.

In the context of logic programs, "unfolding" means [20] to transform a program rule by replacing it by the set of rules obtained after application of a computation step (in all its possible forms) on the body of the selected rule. Unfolding was defined for the multi-adjoint framework in [9].

Let $\mathcal{P}$ be a program and $\mathcal{R} \equiv \langle A \leftarrow \mathcal{B}; v\rangle \in \mathcal{P}$ a program rule. Then, the fuzzy unfolding of program $\mathcal{P}$ with respect to rule $\mathcal{R}$ is the new program $\mathcal{P}' = (\mathcal{P} \smallsetminus \{\mathcal{R}\}) \cup \mathcal{U}$ such that: $\mathcal{U} = \{\langle A\sigma \leftarrow \mathcal{B}'; v\rangle \mid \langle \mathcal{B}; id\rangle {\leadsto_{AS}} \langle \mathcal{B}'; \sigma\rangle\}$. Note that the set $\mathcal{U}$ may be a singleton when the unfolding step is performed on an atom of the body with a predicate at the root which is defined deterministically by just one rule. Unfolding is a program transformation technique which preserve semantics.

Proposition 3. *Let $\mathcal{P}$ be a multi-adjoint program. Any G-reductant of $\mathcal{P}$ can be transformed into a maximal critical reductant of $\mathcal{P}$ after a sequence of unfolding steps.*

Proof. Without loss of generality, we first analyze the case of a program $\mathcal{P}$ with a pair of rules, $\{\langle p(t_{11}, \dots, t_{1m}){\leftarrow_1}\ \mathcal{B}_1; v_1\rangle, \langle p(t_{21}, \dots, t_{2m}){\leftarrow_2}\ \mathcal{B}_2; v_2\rangle\}$ which are the rules defining a predicate p.

By Definition 6 the program $\mathcal{P}$ is extended with a rule $R_{\approx} \equiv \langle X \approx X; \top\rangle$ defining the unification operator, $\approx$, and it is possible to build the G-reductant

$$\mathcal{R}^0 \equiv \langle p(X_1, \dots, X_m){\leftarrow}\ @(\hat{\theta}_1\&\mathcal{B}_1, \hat{\theta}_2\&\mathcal{B}_2); \top\rangle,$$

where $\hat{\theta}_i \equiv (X_1 \approx t_{i1} \& \ldots \& X_m \approx t_{im})$, $i \in \{1, 2\}$ and @ is given as in Definition 6.

Note that, when a program $\mathcal{P}$ is extended with a rule $R_{\approx} \equiv \langle X \approx X; \top\rangle$ defining the unification operator, $\approx$, a unification process can be simulated by a derivation where all admisible steps are performed using the rule $R_{\approx}$. Now, starting from $\mathcal{R}^0$, acting as the unfolded rule, we can built the following unfolding transformation step with respect to $R_{\approx}$, the unfolding rule, and the subgoal $X_1 \approx t_{11}$. In this case $\langle X_1 \approx t_{11}, id\rangle \leadsto_{AS} \langle \top, \{X_1/t_{11}\}\rangle$ and we obtain:

$$\begin{aligned}\mathcal{R}^1 \equiv \langle p(t_{11}, \ldots, X_m) \leftarrow\ & @((\top \& X_2 \approx t_{11} \& \ldots \& X_m \approx t_{1m}) \& \mathcal{B}_1, \\ & (t_{11} \approx t_{21} \& \ldots \& X_m \approx t_{2m}) \& \mathcal{B}_2); \top\rangle\end{aligned}$$

Hence, repeating a sequence of unfolding steps on $\mathcal{R}^{i-1}$ with respect to $R_{\approx}$ and the subgoal $X_i \approx t_{i1}$, for $i \in \{2, \ldots, m\}$, since $\langle X_1 \approx t_{1i}, id\rangle \leadsto_{AS} \langle \top, \{X_i/t_{1i}\}\rangle$, we obtain:

$$\begin{aligned}\mathcal{R}^m \equiv \langle p(t_{11}, \ldots, t_{1m}) \leftarrow\ & @((\top \& \ldots \& \top) \& \mathcal{B}_1, \\ & (t_{11} \approx t_{21} \& \ldots \& t_{1m} \approx t_{2m}) \& \mathcal{B}_2); \top\rangle,\end{aligned}$$

Doing some simplifications by applying interpretative steps:

$$\mathcal{R}^{m+1} \equiv \langle p(t_{11}, \ldots, t_{1m}) \leftarrow @(\mathcal{B}_1, (t_{11} \approx t_{21} \& \ldots \& t_{1m} \approx t_{2m}) \& \mathcal{B}_2); \top\rangle,$$

At this point, we consider two cases:

- The rules in $\mathcal{P}$ are critical rules: In this case, by definition, the atoms at the head of the rules are unifiable, that is, $\theta = mgu\{p(t_{11}, \ldots, t_{1m}), p(t_{21}, \ldots, t_{2m})\} \not\equiv fail$. Therefore, is possible the following unfolding sequence of the subgoal $t_{11} \approx t_{21} \& \ldots \& t_{1m} \approx t_{2m}$ of the body:

$$\langle t_{11} \approx t_{21} \& \ldots \& t_{1m} \approx t_{2m}, id\rangle \leadsto^\star \langle \top, \theta\rangle.$$

  Then, we can perform the following unfolding transformation of $\mathcal{R}^{m+1}$:

$$\begin{aligned}\mathcal{R}^{m+2} &\equiv \langle p(t_{11}, \ldots, t_{1m})\theta \leftarrow @(\mathcal{B}_1, \top \& \mathcal{B}_2)\theta; \top\rangle, \\ &= \langle p(t_{11}, \ldots, t_{1m})\theta \leftarrow @(\mathcal{B}_1, \mathcal{B}_2)\theta; \top\rangle \\ &= \langle p(t_{11}, \ldots, t_{1m})\theta \leftarrow @_{\text{sup}}(v_1 \&_1 \mathcal{B}_1, v_2 \&_2 \mathcal{B}_2; \top\rangle\end{aligned}$$

  On the other hand, because the rules form a critical set of rules, it is possible to build the maximal critical reductant:

$$\mathcal{R} \equiv \langle p(t_{11}, \ldots, t_{1m})\theta \leftarrow @_{sup}(v_1 \&_1 \mathcal{B}_1, v_2 \&_2 \mathcal{B}_2)\theta; \top\rangle.$$

  which clearly is equivalent to $\mathcal{R}^{m+2}$.
- The rules in $\mathcal{P}$ are not critical rules: In this case, the terms of some subgoal $t_{1i} \approx t_{2i}$, with $i \in \{1, \ldots, m\}$, cannot be unified and so, $\langle t_{1i} \approx t_{2i}, id\rangle \leadsto_{AS} \langle \bot, id\}\rangle$. Therefore:

$$\begin{aligned}\mathcal{R}^{m+2} &\equiv \langle p(t_{11}, \ldots, t_{1m}) \leftarrow @(\mathcal{B}_1, t_{11} \approx t_{21} \& \ldots \& \bot \& \ldots \& t_{1m} \approx t_{2m} \& \mathcal{B}_2); \top\rangle, \\ &= \langle p(t_{11}, \ldots, t_{1m}) \leftarrow @(\mathcal{B}_1, \bot \& \mathcal{B}_2); \top\rangle, \\ &= \langle p(t_{11}, \ldots, t_{1m}) \leftarrow v_1 \&_1 \mathcal{B}_1; \top\rangle \equiv \mathcal{R}_1\end{aligned}$$

  In this case the rules in $\mathcal{P}$ are both maximal critical reductant themself.

Therefore, in both cases the G-reductant of $\mathcal{P}$ is a maximal critical reductant of $\mathcal{P}$, after a sequence of unfolding steps. This result can be easily extended, by recurrence, to programs with a finite arbitrary number of rules.

An important question, given a multi-adjoint program, is to know how many reductants are necessary to take into account in order to preserve the approximate completeness of the framework. We focus now on this question, and drive the discussion by means of one small but significative example.

Example 3. Given the program

$$\mathcal{P} = \{\mathcal{R}_1 : \langle p(a, Y, Z) \leftarrow ; v_1\rangle, \mathcal{R}_2 : \langle p(X, b, Z) \leftarrow ; v_2\rangle, \mathcal{R}_3 : \langle p(X, Y, c) \leftarrow ; v_3\rangle\}$$

one can compute the following reductants, according to the corresponding sets of critical rules:

– Set of critical rules $\{\mathcal{R}_1, \mathcal{R}_2\}$: $Red_1 \equiv \langle p(a, b, Z) \leftarrow sup\{v_1, v_2\}; \top\rangle$, since $mgu\{p(a, Y, Z), p(X, b, Z)\} = \{X/a, Y/b\}$.
– Set of critical rules $\{\mathcal{R}_1, \mathcal{R}_3\}$: $Red_2 \equiv \langle p(a, Y, c) \leftarrow sup\{v_1, v_3\}; \top\rangle$, since $mgu\{p(a, Y, Z), p(X, Y, c)\} = \{X/a, Z/c\}$.
– Set of critical rules $\{\mathcal{R}_2, \mathcal{R}_3\}$: $Red_3 \equiv \langle p(X, b, c) \leftarrow sup\{v_2, v_3\}; \top\rangle$, since $mgu\{p(X, b, Z), p(X, Y, c)\} = \{Y/b, Z/c\}$.
– Set of critical rules $\{\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3\}$: $Red_4 \equiv \langle p(a, b, c) \leftarrow sup\{v_1, v_2, v_3\}; \top\rangle$, since $mgu\{p(a, Y, Z), p(X, b, Z), p(X, Y, c)\} = \{X/a, Y/b, Z/c\}$. Note that this is a maximal reductant.

Some derivations that can be performed with the original program $\mathcal{P}$ and the goal $p(a, b, Z)$ are:

– $\langle p(a, b, Z); id\rangle \overset{\mathcal{R}_1}{\leadsto}_{\text{AS}} \langle v_1; \{Y_1/b, Z/Z_1\}\rangle$ with fuzzy computed answer, fca1, $\langle v_1; \{Z/Z_1\}\rangle$.
– $\langle p(a, b, Z); id\rangle \overset{\mathcal{R}_2}{\leadsto}_{\text{AS}} \langle v_2; \{X_1/a, Z/Z_1\}\rangle$ with fuzzy computed answer, fca2, $\langle v_2; \{Z/Z_1\}\rangle$.
– $\langle p(a, b, Z); id\rangle \overset{\mathcal{R}_3}{\leadsto}_{\text{AS}} \langle v_3; \{X_1/a, Y_1/b, Z/c\}\rangle$ with fuzzy computed answer, fca3, $\langle v_3; \{Z/c\}\rangle$.

It is worth to state that the fuzzy computed answers fca1 and fca2 are problematic, because they compute the same answer substitution but different truth degrees. From this, by soundness, it can be inferred that $\langle v_1; \{Z/Z_1\}\rangle$ and $\langle v_2; \{Z/Z_1\}\rangle$ are correct answers and, therefore, the existence of a correct answer $\langle \sup\{v_1, v_2\}; \{Z/Z_1\}\rangle$ with is better than the preceding correct answers, but not computed by the operational mechanism. In order to solve this problem it is necessary to complete $\mathcal{P}$ with the reductant Red_1. Now it is possible the following derivation: $\langle p(a, b, Z); id\rangle \overset{Red_1}{\leadsto}_{\text{AS}} \langle \sup\{v_1, v_2\}; \{Z/Z_1\}\rangle$, that computes the missing fuzzy computed answer.

On the other hand, note that just including the unique maximal reductant Red_4 (disregarding the other reductants of $\mathcal{P}$) does not solve the problem. The only leading derivation in this case is: $\langle p(a,b,Z); id\rangle \overset{Red_4}{\leadsto}_{\mathrm{AS}} \langle \sup\{v_1,v_2,v_3\}; \{Z/c\}\rangle$, computing the fuzzy computed answer $\langle \sup\{v_1,v_2,v_3\}; \{Z/c\}\rangle$ but not the correct answer $\langle \sup\{v_1,v_2\}; \{Z/Z_1\}\rangle$.

Finally, if we consider the goal $p(a,b,c)$ and the program $\mathcal{P}$, it is possible to obtain the following one step derivations:

– $\langle p(a,b,c); id\rangle \overset{\mathcal{R}_1}{\leadsto}_{\mathrm{AS}} \langle v_1; \{Y_1/b, Z_1/c\}\rangle$ with fuzzy computed answer, fca4, $\langle v_1; id\rangle$.
– $\langle p(a,b,c); id\rangle \overset{\mathcal{R}_2}{\leadsto}_{\mathrm{AS}} \langle v_2; \{X_1/a, Z_1/c\}\rangle$ with fuzzy computed answer, fca5, $\langle v_2; id\rangle$.
– $\langle p(a,b,c); id\rangle \overset{\mathcal{R}_3}{\leadsto}_{\mathrm{AS}} \langle v_3; \{X_1/a, Y_1/b\}\rangle$ with fuzzy computed answer, fca6, $\langle v_3; id\rangle$.

The fuzzy computed answers fca4, fca5 and fca6 are correct answers as well, and this leads, by definition, to the existence of the correct answer $\langle \sup\{v_1,v_2,v_3\}; id\rangle$. This makes necessary the extension of $\mathcal{P}$ with the reductant Red_4 in order to compute it: $\langle p(a,b,c); id\rangle \overset{Red_4}{\leadsto}_{\mathrm{AS}} \langle \sup\{v_1,v_2,v_3\}; id\rangle$, □

The previous example shows that all reductants of a program (associated with the different sets of critical rules) are necessary for preserving the (approximate) completeness of the multi-adjoint logic programming framework. Also, it can be seen as a counter-example to the statement claiming that, "it is only necessary to extend a multi-adjoint program with a significative subset of its reductants to preserve completeness".

6 Computing Reductants

In the last section we justified that a multi-adjoint program has to include all its reductants in order to preserve the approximate completeness property. Here, we describe an efficient algorithm for computing all the reductants associated with a multi-adjoint program.

Our algorithm takes a program $\mathcal{P}$ and, by computing what we call a set of unifiable configurations, it delivers the set of associated reductants. A unifiable configuration is a pair made up of a set of critical rules (that is, rules with unifiable head atoms), jointly with the m.g.u. of their heads.

Definition 9. *Let $\mathcal{P}$ be a multi-adjoint program and let $C = \{\langle H_1 \leftarrow \mathcal{Q}_1; \alpha_1\rangle, \ldots, \langle H_n \leftarrow \mathcal{Q}_n; \alpha_1\rangle\}$ be a set of critical rules of $\mathcal{P}$ with $\theta = mgu\{H_1, \ldots, H_n\}$. Then, the pair $\langle C, \theta\rangle$ is a* unifiable configuration. *We say that a configuration is* initial *if $C = \{\mathcal{R}\}$, with $\mathcal{R} \in \mathcal{P}$, and $\theta = id$. If $C = \varnothing$ and $\theta = id$ we say that the configuration is* null.

Example 4. Given the program of Example 3,

$$\langle \{\langle p(a,Y,Z) \leftarrow; v_1\rangle, \langle p(X,b,Z) \leftarrow; v_2\rangle\}, \{X/a, Y/b\}\rangle$$

is a unifiable configuration, where the substitution $\{X/a, Y/b\}$ is the m.g.u. of $p(a,Y,Z)$ and $p(X,b,Z)$. On the other hand, $\langle\{\langle p(a,Y,Z)\leftarrow; v_1\rangle\}, id\rangle$ is an initial configuration. □

Note that, for a program $\mathcal{P}$ the sets of critical rules belong to the powerset of $\mathcal{P}$. Therefore, the sets of unifiable configurations can be ordered by inclusion and organized in a Hasse diagram. Also, starting from the null configuration we can organize the configurations by levels. If $\langle C, \theta\rangle$ is a configuration in level N, then $|C| = N$. It is noteworthy that the configurations in the level N can be obtained by joining all pairs of configurations in the level $N-1$.

Our algorithm takes the set of initial configurations as a basis to generate the final set of configurations. It repeatedly generates the configuration subsets in the level N (subsets of cardinality N) from the unifiable configuration subsets in the level $N-1$ (subsets of cardinality $N-1$). Concurrently, the subsets of the new level are tested to be unifiable. If they are unifiable the configuration is stored; otherwise, it is discarded. The process continues until either there are not unifiable configuration subsets in the new level N or it is a singleton.

Algorithm 1

```
Input: A Program $\mathcal{P} = \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$.
Output: A set UnifConfs of unifiable configuration of $\mathcal{P}$.
Initialization: UnifConfs := $\emptyset$;
                CurrentLevel := $\{\langle\{\mathcal{R}_i\}, id\rangle \mid 1 \leq i \leq n\}$;
Repeat

    NextLevel := $\emptyset$;
    %% built next level
    Let CurrentLevel := $\{\langle C_i, \theta_i\rangle \mid 1 \leq i \leq k\}$,
    For each $C_i$ and $C_j$, in CurrentLevel, do
      1. built the new configuration NewConf := $\langle C_i \cup C_j, \theta_{ij}\rangle$
         if $\theta_{ij} = mgu(C_i \cup C_j) \not\equiv$ fail;
      2. if NewConf $\not\subseteq$ NextLevel then NextLevel := NextLevel $\cup$ {NewConf};
    endFor
    UnifConfs := UnifConfs $\cup$ NextLevel;
    CurrentLevel := NextLevel;

until CurrentLevel := $\emptyset$ or CurrentLevel is a singleton
Return UnifConfs
```

Once the set of unifiable configurations has been generated, the reductants of $\mathcal{P}$ are built by taken each configuration in UnifConfs and applying Definition 8.

In the rest of the section, we give some notes on the complexity and feasibility of this algorithm.

Assume a program with d definite predicates and an average of n rules defining one of those predicates. Then, for each definite predicate we have a set S formed by n rules and, in the worst case, we have to inspect all the elements (configurations) of the powerset of S, except the bottom element and the singleton elements (initial configurations). If we use a Hasse diagram to order the elements of S by inclusion, a level l of the diagram contains all the subsets of S

formed by the combinations of size l taken from S. Therefore, under the previous assumptions, the number of unification problems to be solved is:

$$d \times \sum_{l=2}^{n} \frac{n!}{(n-l)!\ l!}$$

Computing all the reductants associated with a multi-adjoint program is something feasible due to the programming style in the context of a logic programming language. In this domain area programs a composed by definite predicates whose definitions consist of a reduced number of rules (usually, not more than four or five rules), hence the number of configurations to be inspected is very limited for each definite predicate. Moreover, using a minimally sophisticated algorithm like ours, it is possible to avoid the inspection (and the generation) of a great number of those configurations which are not really unifiable.

On the other hand, an additional thing to be taken into account is that, in the field of logic programming is common to use a pattern matching programming technique, what usually leads to rules whose heads do not unify.

7 Conclusions

We have revisited the concept of a reductant in the framework of multi-adjoint logic programming. After presenting a summary of the different notions of reductant appeared in this field, we have defined the concept of a set of critical rules and a new notion of reductant. Significantly, the new notion of reductant allows for recovering approximate completeness by including just finitely many critical reductants; contrariwise to that happens with the previous notions proposed in the literature, which generate infinitely many of them. We have studied the formal properties of the new notion of reductant and some of its relationships with other notions of reductant. Specifically, we have proved that, as expected, any model of $\mathcal{P}$ is also a model of their critical reductants; however, the converse is not always the case, which only holds under very restrictive conditions, finally, we have proved that any G-reductant can be transformed into a critical reductant by using unfolding transformations.

In the final section, we have shown, by means of small but significative examples, that it is necessary to compute all the reductants associated with a multi-adjoint logic program (and not only its maximal reductants). These reductants must be attached to the program in order to preserve the approximate completeness property of the multi-adjoint logic programming framework. Also we have proposed an efficient algorithm to compute all the reductants of a multi-adjoint logic program.

As future work, more properties of critical reductants have to be studied together with the computation of the minimal model of a program from critical reductants. In addition, the influence of the use of critical reductants in the tabling procedure will also be studied.

References

1. J. F. Baldwin, T. P. Martin, and B. W. Pilsworth. *Fril- Fuzzy and Evidential Reasoning in Artificial Intelligence.* John Wiley &; Sons, Inc., 1995.
2. T. H. Cao and N. V. Noi. A framework for linguistic logic programming. *International Journal of Intelligent Systems*, 25(6):559–580, 2010.
3. P. Eklund, M. Galán, R. Helgesson, J. Kortelainen, G. Moreno, and C. Vázquez. Towards categorical fuzzy logic programming. In F. Masulli, G. Pasi, and R. Yager, editors, *Fuzzy Logic and Applications*, volume 8256 of *Lecture Notes in Computer Science*, pages 109–121. Springer International Publishing, 2013.
4. P. Eklund and F. Klawonn. Neural fuzzy logic programming. *Neural Networks, IEEE Transactions on*, 3(5):815–818, 1992.
5. S. Guadarrama, S. Muñoz, and C. Vaucheret. Fuzzy prolog: A new approach using soft constraints propagation. *Fuzzy Sets and Systems*, 144(1):127–150, 2004.
6. P. Hájek. *Metamathematics of fuzzy logic*, volume 4. Springer, 1998.
7. M. Ishizuka and N. Kanai. Prolog-ELF Incorporating Fuzzy Logic. In A. K. Joshi, editor, *Proc. of the 9th International Joint Conference on Artificial Intelligence (IJCAI'85). Los Angeles, CA, USA*, pages 701–703. Morgan Kaufmann, 1985.
8. N. Jones, C. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation.* Prentice-Hall, Englewood Cliffs, NJ, 1993.
9. P. Julián, G. Moreno, and J. Penabad. On Fuzzy Unfolding. A Multi-adjoint Approach. *Fuzzy Sets and Systems, Elsevier*, 154:16–33, 2005.
10. P. Julián, G. Moreno, and J. Penabad. An Improved Reductant Calculus using Fuzzy Partial Evaluation Techniques. *Fuzzy Sets and Systems*, 160:162–181, 2009.
11. M. Kifer and V. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12:335–367, 1992.
12. S. Krajči, R. Lencses, and P. Vojtáš. A comparison of fuzzy and annotated logic programming. *Fuzzy Sets and Systems*, 144(1):173–192, 2004.
13. T. Kuhr and V. Vychodil. Fuzzy logic programming reduced to reasoning with attribute implications. *Fuzzy Sets and Systems*, 2014. In press.
14. V. H. Le, F. Liu, and D. K. Tran. Fuzzy linguistic logic programming and its applications. *Theory and Practice of Logic Programming*, 9:309–341, 2009.
15. J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Similarity-based unification: a multi-adjoint approach. *Fuzzy Sets and Systems*, 146(1):43–62, 2004.
16. P. Morcillo and G. Moreno. A practical approach for ensuring completeness of multi-adjoint logic computations via general reductants. In G. M. P. Lucio and R. Peña, editors, *Proc. of IX Jornadas sobre Programación y Lenguajes, PROLE'09, San Sebastián, Spain, September 8-11*, pages 355–363. Universidad del País Vasco, 2009. (ISBN 978-84-692-4600-9).
17. P. Morcillo and G. Moreno. Improving completeness in multi-adjoint logic computations via general reductants. In *Proc. of 2011 IEEE Symposium on Foundations of Computational Intelligence, April 11-15, Paris, France*, pages 138–145. IEEE, 2011.
18. P. Morcillo and G. Moreno. Simplifying general reductants with unfolding-based techniques. In P. Arenas, V. Gulías, and P. Nogueira, editors, *Proc. of XI Jornadas sobre Programación y Lenguajes, PROLE'11, A Coruña, Spain, September 5-7*, pages 154–168 (sección de trabajos en progreso). Universidade da Coruña (ISBN 978-84-9749-487-8), 2011.
19. G. Moreno and V. Pascual. A hybrid programming scheme combining fuzzy-logic and functional-logic resources. *Fuzzy Sets and Systems*, 160(10):1402 – 1419, 2009. Special Issue: Fuzzy Sets in Interdisciplinary Perception and Intelligence.

20. A. Pettorossi and M. Proietti. Rules and Strategies for Transforming Functional and Logic Programs. *ACM Computing Surveys*, 28(2):360–414, 1996.
21. P. Vojtáš. Fuzzy Logic Programming. *Fuzzy Sets and Systems*, 124(1):361–370, 2001.
22. H. Yasui, Y. Hamada, and M. Mukaidono. Fuzzy prolog based on Lukasiewicz implication and bounded product. In *Proc. of IEEE Symp on Fuzzy Systems FUZZ-IEEE*, pages 949–954, 1995.