On reductants in the framework of multi-adjoint logic programming[☆]

P. Julián-Iranzo[a], J. Medina[b], M. Ojeda-Aciego[c]

[a] *Dep. of Information Technologies and Systems, Univ. of Castilla-La Mancha, Spain.*
[b] *Dep. de Matemáticas, Universidad de Cádiz, Spain.*
[c] *Universidad de Málaga. Departamento de Matemática Aplicada, Spain.*

Abstract

Reductants are a special kind of fuzzy rules which constitute an essential theoretical tool for proving correctness properties. As it has been reported, when interpreted on a partially ordered structure, a multi-adjoint logic program has to include all its reductants in order to preserve the (approximate) completeness property. After a short survey of the different notions of reductant that have been developed for multi-adjoint logic programs, we introduce a new and more adequate, notion of reductant in the multi-adjoint framework. We study some of its properties and its relationships with other notions of reductants. In addition, we 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

General forms of logic programming, mainly related to fuzzy extensions, can be found in the literature since the late eighties [8, 5, 13, 1, 25]; more recently, some authors introduced some generalizations and studied their interrelationships [2, 22, 16, 14, 6, 24]; nowadays, the field of fuzzy logic programming is still an appealing research topic which is being studied from

[☆] This is an extended version of the work "Revisiting reductants in the multi-adjoint logic programming framework". Lect. Notes in Artificial Intelligence 8761:694–702, 2014

Email addresses: Pascual.Julian@uclm.es (P. Julián-Iranzo), jesus.medina@uca.es (J. Medina), aciego@uma.es (M. Ojeda-Aciego)

different standpoints, see for instance [15, 4, 20]. This paper focuses on the general framework of multi-adjoint logic programming [18] and, specifically, on the most adequate notion of reductant for a multi-adjoint logic program.

The multi-adjoint logic programming (MALP) paradigm is a flexible generalization of logic programming which allows for combining many fuzzy logic-related features within the machinery of logic programming. Roughly speaking, a multi-adjoint logic program can be seen as a set of implicational rules annotated by a truth degree which is an element of a complete residuated lattice. Among the many new features of the multi-adjoint framework, it is worth to remark its flexibility, in that rules need not be written with the same (fuzzy) implication, the most suitable for each rule can be considered instead; a second important feature is that it allows using conjunctors in the body of the rules which are neither commutative or associative.

To the best of our knowledge, the notion of reductant was firstly introduced in [13], for the so-called generalized annotated logic programs, as a tool to deal with problems related to incompleteness. The multi-adjoint paradigm has to face similar problems related to incompleteness, which arise specially when one interprets programs over a non-linear lattice. Specifically, incomparable elements in such a lattice allow for constructing programs for which it is not possible to compute the greatest correct answer, see [18]; therefore, should we wish to preserve the approximate completeness property, the resulting multi-adjoint programs will have to include all the reductants, but this would dramatically increase the difficulty of implementing *efficient* programming environments for the multi-adjoint paradigm.

A common problem of the first approaches to the notion of reductant in this field is that, very often, infinitely many reductants are required to be considered in order to guarantee approximate completeness for some programs. Thus, in order 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 surveying the previous notions of reductant already available for multi-adjoint programs, we define a new and more adequate notion of reductant inspired on the one proposed by Kifer and Subrahmanian in the context of generalized annotated logic programs [13]. We study some of its formal properties and we give an efficient algorithm for computing all the reductants associated to a multi-adjoint logic program.

As stated before, this paper is an extended and improved version of [10] and, necessarily, has the same structure (note that, in the following descrip-

tion of the content of this manuscript, the differences and extensions with respect to the conference paper are explicitly *emphasized*): in Section 2, we summarize the syntax and semantics of multi-adjoint logic programs; then, in Section 3, we recall different notions of reductants (the original one, G-reductants and PE-reductants — *the latter were not included in* [10]) along with their basic properties; in Section 4, the new definition of critical reductant is presented; then, in Section 5 its formal properties are stated (*together with their proofs*). The rest of the paper is completely new, that is, *Sections 5.1, 5.2, 5.3 and 6 form a proper extension of* [10]: specifically, Section 5.1 includes a detailed comparison between the new notion of critical reductant and PE-reductants (and, hence, with the original reductants as well), whereas in Section 5.2 we compare critical reductants with G-reductants (in particular, we prove that G-reductants do not preserve approximate completeness). Then, in Section 5.3, we consider the problem of how many reductants should be computed in order to guarantee approximate completeness and, then, in Section 6 we describe an efficient algorithm for computing all the reductants associated with a multi-adjoint program. The final section contains some conclusions and prospects for future work.

2. Syntax and Semantics of Multi-adjoint Logic Programs

For the sake of self-completion, the preliminary definitions on the multi-adjoint framework are given in this section.

Hereafter, we will be working with a first order language, $\mathcal{L}$, containing variables, function symbols, predicate symbols, constants, the classical quantifiers ($\forall$ and $\exists$), and several (arbitrary) connectives, which are intended to provide extended expressiveness features to the language.

As usual in a 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 operator subsumes conjunctors (denoted by $\&_k$), disjunctors ($\vee_l$), and hybrid operators. The truth function[1] for an n-ary aggregation operator $@\colon L^n \to L$ is required to be monotone and fulfill $@(\top, \ldots, \top) = \top$, $@(\bot, \ldots, \bot) = \bot$.

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

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 $B_1, \ldots, B_n$, $n \geq 0$, 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* from L, usually assigned by an expert.

Concerning the semantics, formulas will be interpreted on a multi-adjoint lattice, i.e. the underlying set of truth-values is a complete lattice L together with a collection of adjoint pairs intended to reproduce the application of *modus ponens* [7]. In this framework, it is sufficient to consider Herbrand interpretations in order to define a declarative semantics. See [18] 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* [7] that, given an atomic goal A and a program rule $\langle H \leftarrow_i \mathcal{B}; v \rangle$ for which 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 2.1 (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 $\rightsquigarrow_{AS} \subseteq (\mathcal{E} \times \mathcal{E})$ is the smallest relation satisfying the following* admissible rules *(where we consider that A is the selected atom in $\mathcal{Q}$):*

1) $\langle \mathcal{Q}[A]; \sigma \rangle \rightsquigarrow_{AS} \langle (\mathcal{Q}[A/v \&_i \mathcal{B}])\theta; \sigma\theta \rangle$ *if* $\theta = mgu(\{A = H\})$, $\langle H \leftarrow_i \mathcal{B}; v \rangle$ *in* $\mathcal{P}$.

2) $\langle \mathcal{Q}[A]; \sigma \rangle \rightsquigarrow_{AS} \langle (\mathcal{Q}[A/\bot]); \sigma \rangle$ *if there is no rule in* $\mathcal{P}$ *whose head unifies with* A.

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

Definition 2.2. *Let* $\mathcal{P}$ *be a program and let* $\mathcal{Q}$ *be a goal. An* admissible derivation *is a sequence* $\langle \mathcal{Q}; id \rangle \rightsquigarrow^{*}_{AS} \langle \mathcal{Q}'; \theta \rangle$. *When* $\mathcal{Q}'$ *is a formula not containing atoms (i.e., containing truth values instead), 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, then, can be directly interpreted in the multi-adjoint lattice L.

Definition 2.3 (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* $\rightsquigarrow_{IS} \subseteq (\mathcal{E} \times \mathcal{E})$ *is the smallest one satisfying:* $\langle Q[@(r_1, \ldots, r_n)]; \sigma \rangle \rightsquigarrow_{IS} \langle Q[@(r_1, \ldots, r_n)/v]; \sigma \rangle$, *where* v *is the truth value obtained after evaluating* $@(r_1, \ldots, r_n)$ *in the lattice* $\langle L, \preceq \rangle$ *associated with* $\mathcal{P}$.

Definition 2.4. *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* $\langle Q; \sigma \rangle \rightsquigarrow^{*}_{IS} \langle Q'; \sigma \rangle$. *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 to 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.".

Incluir ejemplo explicativo

3. On the notion of reductant for MALP: state of the art

In this section we survey the different notions of reductants raised over the last years in the field of fuzzy logic programming, specially those extended to MALP, 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 [13], and was initially adapted to the multi-adjoint logic programming framework in the following terms [18]:

Definition 3.1 (Reductant). *Given a program $\mathcal{P}$, a ground atom A, and a (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 this definition of reductant is linked with a program and a ground atom, contrary to the original definition of reductant given in [13], which is uniquely linked with 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, to compute all the reductants associated with a program, one has to take into account all the atoms in the Herbrand base of that program, which might be infinitely many. Hence, although this notion of reductant is theoretically valuable it may easily turn impractical because of its potential non-termination. Also, as we have just commented, reductants introduce efficiency penalties when a "completed" program is executed. 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 that can be executed more efficiently.

The second chapter in this story was the attempt to construct an improved version of the previous notion of reductant (Definition 3.1) which, while still preserving the essential semantic properties, can be more efficiently executed. In [12], we showed how such a refined notion of reductant can be built by using Partial Evaluation techniques [9]. In a nutshell, the idea was to use

an arbitrary unfolding tree[2] τ for a program $\mathcal{P}$ and a ground atom A, to construct a new notion of a reductant, called *PE-reductant* for A in $\mathcal{P}$. Intuitively, a PE-reductant is constructed by

1. Generating an unfolding tree τ for $\mathcal{P}$ and A, that is, the tree obtained by unfolding (as much as possible) the atom A in the program;
2. Collecting the set of leaves $S = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ in τ; and
3. Constructing the rule $\langle A \leftarrow @_{\sup}(\mathcal{D}_1, \ldots, \mathcal{D}_n); \top \rangle$, which is the PE-reductant of A in $\mathcal{P}$ with regard to τ.

Formally, we have:

Definition 3.2 (PE-reductant). *Let $\mathcal{P}$ be a program, A a ground atom, and τ an unfolding tree for A in P. A PE-reductant for A in $\mathcal{P}$ with respect to τ, is a rule $\langle A \leftarrow @_{\sup}(\mathcal{D}_1, \ldots, \mathcal{D}_n); \top \rangle$, where the truth function for the intended aggregator $@_{\sup}$ is defined as $@_{\sup}(d_1, \ldots, d_n) = \sup\{d_1, \ldots, d_n\}$, and $\mathcal{D}_1, \ldots, \mathcal{D}_n$ are, respectively, the leaves of τ.*

Notice that the definition of PE-reductant assumes an extended syntax for our language where truth degrees and adjoint conjunctions are allowed in the body of program rules. PE-reductants incorporate information about all the relevant aspects of the rules $\langle C_i \leftarrow_i \mathcal{B}_i; v_i \rangle$ used for the evaluation of the atom A: the truth degree v_i, the adjoint implication and conjunction operators, the computed substitutions and the instances of the bodies $\mathcal{B}_i$. Also, note that, for readability reasons, in the sequel we will name those PE-reductants which are obtained from unfolding trees of depth k, PE^k-reductants. The semantical equivalence between reductants of Definition 3.1 and PE-reductants was established in [12].

Despite PE-reductants may improve the efficiency of multi-adjoint computations they do not overcome the need to compute, sometimes, infinitely many reductants, since they are still linked to the Herbrand base of ground atoms.

As a further step in the path of trying to avoid the proliferation reductants, the new notion of G-reductant was introduced in [19, 20]. The aim was

[2] An *unfolding tree* for a program $\mathcal{P}$ and a goal $\mathcal{Q}$ is a possibly incomplete search tree whose branches are derivations of $\mathcal{Q}$ in $\mathcal{P}$ interleaving admissible and interpretative steps. An unfolding tree may contain unevaluated leaves where no atom (or interpretable expression) has been selected for a further unfolding step, ending derivations at any adequate point.

that a single generalized reductant was required to cover all the (infinitely many) possible calls to atoms headed by a specific predicate symbol defined in a program.

Definition 3.3 (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*

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

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 $p(X_1, \ldots, X_m)$ via the substitution $\theta_i = \{X_1/t_{i1}, \ldots, X_m/t_{im}\}$;*
- $\hat{\theta}_i = (X_1 \approx t_{i1} \& \cdots \& X_m \approx t_{im})$ *with $\approx$ being a unification operator defined by the fact $R_{\approx} = \langle X \approx X; \top \rangle$, 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 3.1.*

A common way of quantifying the efficiency of a computation is counting its number of steps. Notice 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, some extra steps are necessary when using G-reductants in a computation. As a result, computing with this kind of reductants becomes inefficient. This is why unfolding-based techniques were applied in [21] 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.

REWRITE PARAGRAPH Despite the accomplishments obtained by these transformation techniques, the overall process is far from being intuitive and, what is worst, it does not guarantee the approximate completeness of a multi-adjoint logic programming framework (as shown by Proposition 5.10 together with Example 3).

4. A new notion of reductant: sets of critical rules

The new notion of reductant, once again in the line of [13], was introduced in [10] with the aim at solving the aforementioned problems inherent to the other notions of reductant, seeking a new notion of reductant satisfying that:

1. It is not attached to a certain kind of goals for its computation,
2. It 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: $\langle v_1; \theta\rangle, \ldots, \langle v_n; \theta\rangle$. In this case, $\langle \sup\{v_1, \ldots, v_n\}; \theta\rangle$ can be proven to be a correct answer which is not computed by the operational mechanism. INCLUIR CORRECT ANSWER EN PRELIMINARES

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. In order to simplify the discussion, we analyze the situation of a program containing just two facts $\mathcal{R}_1 = \langle H_1, a\rangle$, $\mathcal{R}_2 = \langle H_2, b\rangle$, 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 must be possible to perform admissible steps both with $\mathcal{R}_1$ and $\mathcal{R}_2$ computing the fuzzy computed answers $\langle a; \theta\rangle$ and $\langle b; \theta\rangle$ respectively. But this is only possible when the heads of $\mathcal{R}_1$ and $\mathcal{R}_2$ unify with A: $A\theta = H_1\theta$ and $A\theta = H_2\theta$, which leads to the condition that the heads of $\mathcal{R}_1$ and $\mathcal{R}_2$ should also unify.[3] This is an interesting observation that gives a criterion to decide when a set of rules may cause the problem. In general, this problem may occur when, in our program, sets of rules exist whose heads unify. We shall say that such sets are "sets of critical rules".

Definition 4.1 (Critical Rules). *Let $\mathcal{P}$ be a program, and $\mathcal{R}_1 = \langle H_1 \leftarrow \mathcal{B}_1, v_1\rangle$, and $\mathcal{R}_2 = \langle H_2 \leftarrow \mathcal{B}_2, v_1\rangle$ two rules in $\mathcal{P}$ that are renamed-apart. The rules*

[3] Recall that, if H_1 and H_2 unify, they have a most general unifier.

$\mathcal{R}_1$ *and* $\mathcal{R}_2$ *are said to be* critical *iff* H_1 *and* H_2 *unify, that is, there exists a substitution* $\theta = mgu\{H_1, H_2\} \neq fail$.

A set of rules in $\mathcal{P}$ *which are renamed-apart is said to be 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* $mgu\{p(a, g(Z)), p(Y, g(Y))\} = \{Y/a, Z/a\} \neq fail$. □

The new notion of reductant is introduced in the following definition:

Definition 4.2 (Critical Reductant). *Given a program* $\mathcal{P}$ *and a set of critical rules in* $\mathcal{P}$, $\{\langle H_i \leftarrow_i \mathcal{B}_i; v_i\rangle \mid 1 \leq i \leq n\}$, *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, namely* $\langle p(a, g(a)) \leftarrow \sup\{v_1, v_2\}; \top\rangle$, *which is obtained from the maximal set of critical rules* $\{\mathcal{R}_1, \mathcal{R}_2\}$. *Therefore, this is a maximal reductant.*

Notice, finally, that a set of critical rules can be formed by a program rule $R_1 = \langle H_1 \leftarrow_1 \mathcal{B}_1; v_1\rangle$ and a variant of R_1, say $R_2 = \langle H_2 \leftarrow_1 \mathcal{B}_2; v_1\rangle$. Then, certainly there exists the $mgu\{H_1, H_2\} = \rho$, where ρ is just a renaming, $H_1 = H_1\rho = H_2\rho$ and $\mathcal{B}_1 = \mathcal{B}_1\rho = \mathcal{B}_2\rho$. Hence, the critical reductant of this pair of variant rules is:

$$\langle H_1\rho \leftarrow_1 @_{\sup}(v_1 \&_1 \mathcal{B}_1, v_1 \&_1 \mathcal{B}_2)\rho; \top\rangle =$$
$$\langle H_1\rho \leftarrow_1 @_{\sup}(v_1 \&_1 \mathcal{B}_1\rho, v_1 \&_1 \mathcal{B}_2\rho); \top\rangle = \langle H_1 \leftarrow_1 @_{\sup}(v_1 \&_1 \mathcal{B}_1, v_1 \&_1 \mathcal{B}_1); \top\rangle =$$
$$\langle H_1 \leftarrow_1 v_1 \&_1 \mathcal{B}_1; \top\rangle$$

which, from a semantical point of view, is essentially the program rule R_1. Therefore, each program rule can be considered as a reductant of itself.

5. Formal properties of critical reductants

Some important properties of critical reductants, which are substantive for the correctness of the multi-adjoint logic programming framework, are stated and formally proved in this section.

The first result is a straightforward lemma, which will be used later.

Lemma 5.1. *Let $\mathcal{A}$ be a formula, $\mathcal{I}$ an interpretation and θ a substitution. Then $\mathcal{I}(\mathcal{A}) \leq \mathcal{I}(\mathcal{A}\theta)$.*

The previous lemma is used to prove that reductants are true in every model of a program. Specifically, we have the following proposition:

Proposition 5.2. *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.

To begin with, note that by definition of critical reductant (Definition 4.2) 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

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

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 [18]), we have $v_i \leq \mathcal{I}(H_i \leftarrow_i \mathcal{B}_i)$ and, by Lemma 5.1, $v_i \leq \mathcal{I}(H_i\theta \leftarrow_i \mathcal{B}_i\theta)$. 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 an 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 residual 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 in Proposition 5.3.

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 $\mathcal{P}_R$.

¿Se puede relajar la condición poniendo $\geq$ en lugar de $=$?

Proposition 5.3. *Given a multi-adjoint program $\mathcal{P}$ and a model $\mathcal{I}$ of $\mathcal{P}_R$, if $\mathcal{I}(A) = \inf\{\mathcal{I}(A\theta) \mid \langle A\theta \leftarrow @_{\sup}(v_1 \&_1 \mathcal{B}_1, \ldots, v_n \&_n \mathcal{B}_n)\theta; \top\rangle$ 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 $\mathcal{P}_R$, it is enough 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

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

the inequality $\mathcal{I}(@_{\sup}(v_1 \&_1 \mathcal{B}_1, \ldots, v_n \&_n \mathcal{B}_n)\theta) \leq \mathcal{I}(A\theta)$ holds and so, by Lemma 5.1, we have:

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

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

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

□

In the rest of this section, we study the formal relation between the critical reductants and previous notions of reductant develop for MALP.

5.1. Critical reductants vs original reductants and PE-reductants

We begin this section by studying the relation between critical reductants and the notion of reductant given in [18], which we call *original reductants*. In order to establish this relation, our first step is a proposition which links critical reductants with PE^1-reductants, a particular case of PE-reductants (see Definition 3.2) based on one-step unfolding trees. This way, the notion of PE-reductant serves as an intermediary in that relation.

Proposition 5.4. *Given a multi-adjoint program $\mathcal{P}$, every PE^1-reductant is an instance of a critical reductant of $\mathcal{P}$.*

Proof. Let A be a ground atom and let

$$Red = \langle A \leftarrow @_{\sup}(v_1 \&_1 \mathcal{B}_1\theta_1, \ldots, v_n \&_1 \mathcal{B}_n\theta_n); \top \rangle$$

be the PE^1-reductant for $\mathcal{P}$ and A, obtained from an unfolding tree of depth one, where each substitution θ_i is a unifier of A and the head of a rule $\mathcal{R}_i = \langle H_i \leftarrow_i \mathcal{B}_i; v_i \rangle$ in $\mathcal{P}$. Then it is possible to prove that the set of rules $\mathcal{R}_i$ used in the construction of the PE^1-reductant $\mathcal{R}$ is a set of critical rules.

First note that the rules $H_i \leftarrow_i \mathcal{B}_i$ whose head match with A, can be considered to be standardized apart. Moreover, the atom A is ground by Definition 3.2. Therefore, the substitutions θ_i, such that $A = H_i\theta_i$, do not share variables in common either in their domains or in their ranges. Hence, $\theta = \theta_1\theta_2 \cdots \theta_n = \theta_1 \cup \theta_2 \cup \cdots \cup \theta_n$ and $H_i\theta = H_i\theta_i = A$, for all i, $1 \leq i \leq n$. So, $H_i\theta = H_j\theta$, for each i and j. That is, $\Omega = \{H_i \mid 1 \leq i \leq n\}$ is a unifiable set and, hence, the unification theorem guarantees the existence of a most general unifier, say σ, of Ω. As a result, the set $\{H_i \leftarrow_i \mathcal{B}_i \mid 1 \leq i \leq n\}$ is a set of critical rules.

Now, by Definition 4.2, it is possible to build the critical reductant $Red' = \langle H_1\sigma \leftarrow @_{\sup}(v_1 \&_1 \mathcal{B}_1, \ldots, v_n \&_n \mathcal{B}_n)\sigma; \top \rangle$ of $\mathcal{P}$. Moreover, by definition of most general unifier, one has $\sigma \leq \theta$ and, therefore, Red is an instance of Red'. □

The proposition above states that every PE^1-reductant Red obtained using Definition 3.2 is covered by a critical reductant Red', which is more general than the first one. Therefore, every admissible step performed using Red can be reproduced by applying the critical reductant Red'. The precise relation between these two kinds of steps is established in the following lemma.

Lemma 5.5. *Let $\mathcal{P}$ be a program, $\mathcal{G}$ be a goal and θ a substitution. If there exists a PE^1-reductant Red which performs the admissible step $\langle \mathcal{G}; id\rangle \overset{Red}{\rightsquigarrow}_{AS} \langle \mathcal{Q}\theta; \sigma\rangle$, then there exists a critical reductant Red' which performs the admissible step $\langle \mathcal{G}; id\rangle \overset{Red'}{\rightsquigarrow}_{AS} \langle \mathcal{Q}; \sigma'\rangle$ where $\sigma = \sigma'\theta$ (that is, $\sigma' \leq \sigma$).*

PROOF. If Red is a PE^1-reductant, by definition, we have

$$Red = \langle A \leftarrow @_{\text{sup}}(v_1 \&_1 \mathcal{B}_1, \ldots, v_n \&_1 \mathcal{B}_n)\delta; \top\rangle$$

where A is a ground atom that matches with the heads of a set of rules in $\mathcal{P}$, say $\Gamma \equiv \{H_i \leftarrow_i \mathcal{B}_i \mid 1 \leq i \leq n\}$. This means that $A = H_i\delta$ for all i, $1 \leq i \leq n$, being δ a substitution that does not share variables with goal $\mathcal{G}$. As shown in the proof of Proposition 5.4, Γ is a set of critical rules with m.g.u. $\gamma \leq \delta$ (that is, $\delta = \gamma\lambda$, for some substitution λ). Therefore, there exists a critical reductant $Red' \equiv \langle H_i\gamma \leftarrow @_{\text{sup}}(v_1 \&_1 \mathcal{B}_1, \ldots, v_n \&_1 \mathcal{B}_n)\gamma; \top\rangle$ that covers Red.

Moreover, if it is possible to perform an admissible step for $\mathcal{G}$ and Red, it is because there exists a selected atom of $\mathcal{G}$, say A', such that $A'\sigma = A\sigma$, where $\sigma = mgu(A', A)$. So, $A'\sigma = A\sigma = H_i\delta\sigma = H_i\gamma\lambda\sigma$. Once again, note that γ and λ do not share variables with $\mathcal{G}$ (or particularly A'), since the critical rules in Γ are standardized apart. Therefore, the application of these substitutions to either $\mathcal{G}$ or A', does not change them. Hence $A'\lambda\sigma = A'\sigma = H_i\gamma\lambda\sigma$ and the substitution $\lambda\sigma$ is a unifier of A' and $H_i\gamma$. Now, by the unification theorem, there exists a m.g.u. σ' of A' and $H_i\gamma$, such that $\sigma' \leq \lambda\sigma$, which confirms that it is possible to apply an admissible step with $\mathcal{G}$ and Red'.

More precisely, we have proved that if there exists an admisible step

$$\langle \mathcal{G}[A']; id\rangle \overset{Red}{\rightsquigarrow}_{\text{AS}} \langle \mathcal{G}[@_{\text{sup}}(v_1 \&_1 \mathcal{B}_1, \ldots, v_n \&_1 \mathcal{B}_n)\delta]\sigma; \sigma\rangle,$$

performed by Red, then, there exists an admissible step

$$\langle \mathcal{G}[A']; id\rangle \overset{Red'}{\rightsquigarrow}_{\text{AS}} \langle \mathcal{G}[@_{\text{sup}}(v_1 \&_1 \mathcal{B}_1, \ldots, v_n \&_1 \mathcal{B}_n)\gamma]\sigma'; \sigma'\rangle,$$

performed by Red'.

DICE EL REVISOR QUE YA ESTARÍA PROBADO. REVISAR Y REESCRIBIR.

I don't understand the significance of the rest of the proof starting from "Recalling ...". Isn't the proof already completed before this line? Also, the line "Recalling that..." is not a gramatically correct sentence.

Recalling that δ and γ do not share variables with $\mathcal{G}$ and defining $\mathcal{Q}$ by

$$\mathcal{Q} \equiv \mathcal{G}[@_{\sup}(v_1 \&_1 \mathcal{B}_1, \ldots, v_n \&_1 \mathcal{B}_n)\gamma]\sigma' = \mathcal{G}[@_{\sup}(v_1 \&_1 \mathcal{B}_1, \ldots, v_n \&_1 \mathcal{B}_n)]\gamma\sigma'.$$

Now,

$$\mathcal{G}[@_{\sup}(v_1 \&_1 \mathcal{B}_1, \ldots, v_n \&_1 \mathcal{B}_n)\delta]\sigma = \mathcal{G}[@_{\sup}(v_1 \&_1 \mathcal{B}_1, \ldots, v_n \&_1 \mathcal{B}_n)]\delta\sigma = \mathcal{G}[@_{\sup}(v_1 \&_1 \mathcal{B}_1, \ldots, v_n \&_1 \mathcal{B}_n)]\gamma\lambda\sigma = \mathcal{G}[@_{\sup}(v_1 \&_1 \mathcal{B}_1, \ldots, v_n \&_1 \mathcal{B}_n)]\gamma\sigma'\theta = \mathcal{Q}\theta$$

since $\sigma' \leq \lambda\sigma$ and, therefore, there exists a substitution θ such that $\lambda\sigma = \sigma'\theta$. □

The next result relates the derivations performed in the completion of a program $\mathcal{P}$ by its PE^1-reductants and the derivations performed in the completion obtained by extending $\mathcal{P}$ with its set of critical reductants but, firstly, we need the following intermediate result.

Lemma 5.6 (Lifting lemma). *Let $\mathcal{P}$ be a program, $\mathcal{G}$ be a goal and θ be a substitution. If there exists a derivation $\langle \mathcal{G}\theta; id\rangle \rightsquigarrow^* \langle v; \sigma\rangle$ then there exists a derivation $\langle \mathcal{G}; id\rangle \rightsquigarrow^* \langle v; \sigma'\rangle$, of the same length, with $\sigma' \leq \theta\sigma$.*

PROOF. Similar to the classical lifting lemma in [17]. □

Proposition 5.7. *Given a multi-adjoint program $\mathcal{P}$, let $\Pi_{\mathcal{P}}$ be the completion of $\mathcal{P}$ with all the PE^1-reductants obtained by using Definition 3.2, and let $\Theta_{\mathcal{P}}$ be the completion of $\mathcal{P}$ with all the critical reductants of $\mathcal{P}$. Let $\mathcal{G}$ be a goal. If there exists a derivation $\langle \mathcal{G}; id\rangle \rightsquigarrow^* \langle v; \theta\rangle$ in $\mathcal{P} \cup \Pi_{\mathcal{P}}$, then there exists a derivation $\langle \mathcal{G}; id\rangle \rightsquigarrow^* \langle v; \sigma\rangle$ in $\mathcal{P} \cup \Theta_{\mathcal{P}}$, with $\sigma \leq \theta[\mathcal{V}ar(\mathcal{G})]$.*

PROOF. By induction on the number n of steps of the derivation $\mathcal{D} \equiv (\langle \mathcal{G}; id\rangle \rightsquigarrow^* \langle v; \theta\rangle)$ in $\mathcal{P} \cup \Pi_{\mathcal{P}}$.

- Base case ($n = 1$): In this case, the derivation $\mathcal{D}$ has the shape $\langle \mathcal{G}; id\rangle \rightsquigarrow_{\text{AS}} \langle v; \theta\rangle$ and $\mathcal{G}$ must be an atom. Excluding the trivial case where the step is performed with a rule in $\mathcal{P}$, assume that the step is performed by using a PE^1-reductant $Red \equiv \langle A \leftarrow v; \top\rangle$ where A is a ground atom matching $\mathcal{G}$, that is $\mathcal{G}\theta = A$. The PE^1-reductant Red must have been constructed from only one rule $\mathcal{R} \equiv \langle H \leftarrow; v\rangle \in \mathcal{P}$, with $A = H\delta$. Recall that the rules are taken standardized apart, the substitutions θ and δ do not share variables, and $\mathcal{D}om(\delta)$ does

not share variables with $\mathcal{G}$. Hence $\theta\delta = \theta \cup \delta$ is a unifier of $\mathcal{G}$ and H (i.e., $\mathcal{G}\theta\delta = H\theta\delta$) and there exist a m.g.u. of $\mathcal{G}$ and H. Consider $\sigma = mgu(\mathcal{G}, H)$, then it is possible to apply the step $\langle \mathcal{G}; id\rangle \rightsquigarrow_{\text{AS}} \langle v; \sigma\rangle$ in $\mathcal{P} \cup \Theta_{\mathcal{P}}$. Since $\sigma \leq \theta\delta$, we have that $\sigma \leq \theta[\mathcal{V}ar(\mathcal{G})]$.

- Inductive case ($n > 1$): In this case the derivation $\mathcal{D}$ has the form

$$\mathcal{D} \equiv (\langle \mathcal{G}; id\rangle \overset{Red}{\rightsquigarrow}_{\text{AS}} \langle \mathcal{G}_1\theta; \sigma_1\rangle \rightsquigarrow^* \langle v; \sigma_1\sigma\rangle) \qquad \text{in } \mathcal{P} \cup \Pi_{\mathcal{P}}$$

As in the base case, we can assume that the first step is performed by a PE^1-reductant Red, since this is the non-trivial instance. Now, applying the lifting lemma, it is possible to construct the derivation

$$\mathcal{D}'' \equiv (\langle \mathcal{G}_1; id\rangle \rightsquigarrow^* \langle v; \sigma''\rangle) \qquad \text{in } \mathcal{P} \cup \Pi_{\mathcal{P}}$$

with $\sigma'' \leq \theta\sigma$ and the same number of steps $(n-1)$. Moreover, applying the induction hypothesis, since $\mathcal{D}''$ has $n-1$ steps, there must exist a derivation

$$\mathcal{D}''' \equiv (\langle \mathcal{G}_1; id\rangle \rightsquigarrow^* \langle v; \sigma'\rangle) \qquad \text{in } \mathcal{P} \cup \Theta_{\mathcal{P}}$$

with $\sigma' \leq \sigma''[\mathcal{V}ar(\mathcal{G}_1)]$. Moreover, by Proposition 5.4 and Lemma 5.5, there exists a critical reductant, Red' that covers Red and it is possible to perform the admissible step $\langle \mathcal{G}; id\rangle \overset{Red'}{\rightsquigarrow}_{\text{AS}} \langle \mathcal{G}_1; \sigma_1'\rangle$ with $\sigma_1' \leq \sigma_1$ (more precisely, $\sigma_1 = \sigma_1'\theta$). Therefore, it is possible to merge the last step with derivation $\mathcal{D}'''$ in order to build the following derivation

$$\mathcal{D}' \equiv (\langle \mathcal{G}; id\rangle \overset{Red'}{\rightsquigarrow}_{\text{AS}} \langle \mathcal{G}_1; \sigma_1'\rangle \rightsquigarrow^* \langle v; \sigma_1'\sigma'\rangle) \qquad \text{in } \mathcal{P} \cup \Theta_{\mathcal{P}}$$

where $\sigma_1'\sigma' \leq \sigma_1\sigma'' \leq \sigma_1\theta\sigma = \sigma_1\sigma[\mathcal{V}ar(\mathcal{G})]$, since θ is a ground substitution and $\mathcal{D}om(\theta)$ does not share variables with $\mathcal{G}$ or σ_1. This concludes the proof.

□

The main properties of PE-reductants were established in [12], where it was proved that the original notions of reductant (Definition 3.1) and PE-reductant (Definition 3.2) are both *semantically* and *operationally* equivalent (although the refined notion of PE-reductant is able to increase the efficiency of multi-adjoint logic programs). In particular, a strong equivalence was established with regard to PE^1-reductants, which is summarized in the following theorems:

Theorem 5.8 (See [12], Thm. 13). *Let $\mathcal{P}$ be a program, A a ground atom and $\mathcal{R} \equiv \langle A \leftarrow @(\mathcal{B}_1, \ldots, \mathcal{B}_n)\theta; \top \rangle$ the reductant for A in $\mathcal{P}$, where $\theta = \theta_1 \cdots \theta_n$ and each substitution θ_i is a unifier of A and the head of a rule $\langle C_i \leftarrow_i \mathcal{B}_i; v_i \rangle$. The PE^1-reductant $\mathcal{R}' \equiv \langle A \leftarrow @_{\sup}(v_1 \&_1 \mathcal{B}_1\theta_1, \ldots, v_n \&_n \mathcal{B}_n\theta_n); \top \rangle$, obtained from an unfolding tree of depth one for $\mathcal{P}$ and A, is semantically equivalent to the reductant $\mathcal{R}$, that is, $\mathcal{I}(@(\mathcal{B}_1, \ldots, \mathcal{B}_n)\theta) = \mathcal{I}(@_{\sup}(v_1 \&_1 \mathcal{B}_1\theta_1, \ldots, v_n \&_n \mathcal{B}_n\theta_n))$ for each interpretation $\mathcal{I}$.*

Theorem 5.9 (See [12], Thm. 18). *Let $\mathcal{P}$ be a program, A a ground atom and $\mathcal{R} \equiv \langle A \leftarrow @(\mathcal{B}_1, \ldots, \mathcal{B}_n)\theta; \top \rangle$ the reductant for A in $\mathcal{P}$, where $\theta = \theta_1 \cdots \theta_n$ and each substitution θ_i is a unifier of A and the head of a rule $\langle C_i \leftarrow_i \mathcal{B}_i; v_i \rangle$. The PE^1-reductant $\mathcal{R}' \equiv \langle A \leftarrow @_{\sup}(v_1 \&_1 \mathcal{B}_1\theta_1, \ldots, v_n \&_n \mathcal{B}_n\theta_n); \top \rangle$, obtained from an unfolding tree of depth one for $\mathcal{P}$ and A, is procedurally equivalent to the reductant $\mathcal{R}$, that is, $\mathcal{FCA}(@(\mathcal{B}_1, \ldots, \mathcal{B}_n)\theta) = \mathcal{FCA}(@_{\sup}(v_1 \&_1 \mathcal{B}_1\theta_1, \ldots, v_n \&_n \mathcal{B}_n\theta_n))$.*

In the last theorem $\mathcal{FCA}(E)$ denotes the set of f.c.a.'s provided by a program $\mathcal{P}$ and an expression (goal) E (that is, $\mathcal{FCA}(E) = \{r \in L \mid \langle E; id \rangle \rightsquigarrow^*_{AS/IS} \langle r; \sigma \rangle\}$).

Hence, as a corollary of these results is almost straightforward to adapt the proof of the (approximate) completeness theorem for the multi-adjoint logic programming framework, presented in [18], using critical reductants.

5.2. Critical reductants and G-reductants

Now we turn our attention to the G-reductants studied in [19, 20]. The following proposition relates the notions of critical reductant and G-reductant. 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 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 [23]. Unfolding was defined for the multi-adjoint framework in [11].

Let $\mathcal{P}$ be a program and $\mathcal{R} = \langle A \leftarrow \mathcal{B}; v \rangle \in \mathcal{P}$ a program rule. Then, the fuzzy unfolding of program $\mathcal{P}$ with respect to rule $\mathcal{R}$ is the new program $\mathcal{P}' = (\mathcal{P} \setminus \{\mathcal{R}\}) \cup \mathcal{U}$ in which $\mathcal{U} = \{\langle A\sigma \leftarrow \mathcal{B}'; v \rangle \mid \langle \mathcal{B}; id \rangle \rightsquigarrow_{\text{AS}} \langle \mathcal{B}'; \sigma \rangle\}$. 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 preserves semantics.

Proposition 5.10. *Any G-reductant of a multi-adjoint program $\mathcal{P}$ 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 defining a predicate p, say

$$\{\langle p(t_{11},\ldots,t_{1m}) \leftarrow_1 \mathcal{B}_1; v_1\rangle, \langle p(t_{21},\ldots,t_{2m}) \leftarrow_2 \mathcal{B}_2; v_2\rangle\}.$$

By Definition 3.3, the program $\mathcal{P}$ is extended with the fact $R_{\approx} = \langle X \approx X; \top\rangle$, and the following G-reductant is constructed

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

where $\hat{\theta}_i = (X_1 \approx t_{i1} \& \ldots \& X_m \approx t_{im})$ for $i \in \{1,2\}$, and $@$ is given as in Definitions 3.1 and 3.3.

Recall that, when a program $\mathcal{P}$ is extended with the unification operator $\approx$ (defined by the fact $R_{\approx} \equiv \langle X \approx X; \top\rangle$), any 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 build the following unfolding transformation step with respect to $R_{\approx}$, the unfolding rule, and the subgoal $X_1 \approx t_{11}$. In this case $\langle X_1 \approx t_{11}, id\rangle \rightsquigarrow_{\text{AS}} \langle \top, \{X_1/t_{11}\}\rangle$ and we obtain:

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

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

$$\langle X_1 \approx t_{1i}, id\rangle \rightsquigarrow_{\text{AS}} \langle \top, \{X_i/t_{1i}\}\rangle,$$

we obtain:

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

Doing some simplifications by applying interpretive steps:

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

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, $\theta = mgu\{p(t_{11},\ldots,t_{1m}), p(t_{21},\ldots,t_{2m})\} \neq fail$. Hence, the following unfolding sequence of the subgoal $(t_{11} \approx t_{21} \& \ldots \& t_{1m} \approx t_{2m})$ in the body of $\mathcal{R}^{m+1}$ is possible:

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

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

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

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

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

  which is clearly 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, $\langle t_{1i} \approx t_{2i}, id \rangle \rightsquigarrow_{\text{AS}} \langle \bot, id \} \rangle$. Therefore:

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

  But note that each rule in $\mathcal{P}$ is by itself a maximal critical reductant.

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 finitely many rules. □

The proposition above admits two different readings:

1. *Improved* G-reductants (that is, G-reductants improved by means of unfolding transformations) are a subset of critical reductants; they are equivalent to maximal critical reductants;
2. G-reductants, per se, are less efficient than the corresponding maximal critical reductants and they need to be improved before they are used.

MEJOR COMO SUBSECCIÓN DE LA ANTERIOR

5.3. Problems related to the computation of reductants

In this section, we will firstly focus on the problem of estimating how many reductants one has to take into account in a multi-adjoint program in order to preserve the approximate completeness of the framework. The discussion will be driven by means of a small but significant 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* $\{\mathcal{R}_1, \mathcal{R}_2\}$*:* $Red_1 \equiv \langle p(a, b, Z) \leftarrow \sup\{v_1, v_2\}; \top\rangle$,
- *Set of critical rules* $\{\mathcal{R}_1, \mathcal{R}_3\}$*:* $Red_2 \equiv \langle p(a, Y, c) \leftarrow \sup\{v_1, v_3\}; \top\rangle$,
- *Set of critical rules* $\{\mathcal{R}_2, \mathcal{R}_3\}$*:* $Red_3 \equiv \langle p(X, b, c) \leftarrow \sup\{v_2, v_3\}; \top\rangle$,
- *Set of critical rules* $\{\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3\}$*:* $Red_4 \equiv \langle p(a, b, c) \leftarrow \sup\{v_1, v_2, v_3\}; \top\rangle$. *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:*

- $\langle p(a, b, Z); id\rangle \overset{\mathcal{R}_1}{\leadsto}_{AS} \langle v_1; \{Y_1/b, Z/Z_1\}\rangle$ *with f.c.a., fca1,* $\langle v_1; \{Z/Z_1\}\rangle$,
- $\langle p(a, b, Z); id\rangle \overset{\mathcal{R}_2}{\leadsto}_{AS} \langle v_2; \{X_1/a, Z/Z_1\}\rangle$ *with f.c.a., fca2,* $\langle v_2; \{Z/Z_1\}\rangle$,
- $\langle p(a, b, Z); id\rangle \overset{\mathcal{R}_3}{\leadsto}_{AS} \langle v_3; \{X_1/a, Y_1/b, Z/c\}\rangle$ *with f.c.a., 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 with different truth degrees. From this, by soundness theorem, it can be inferred that $\langle v_1; \{Z/Z_1\}\rangle$ *and* $\langle v_2; \{Z/Z_1\}\rangle$ *are both correct answers and, therefore, the correct answer* $\langle \sup\{v_1, v_2\}; \{Z/Z_1\}\rangle$*, which is better than the preceding correct answers, cannot be computed by the operational mechanism. In order to solve this problem it is necessary to complete* $\mathcal{P}$ *with the reductant* Red_1*. Now the following derivation computes the missing fuzzy computed answer*

$$\langle p(a, b, Z); id\rangle \overset{Red_1}{\rightsquigarrow}_{AS} \langle \sup\{v_1, v_2\}; \{Z/Z_1\}\rangle.$$

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:* $\langle p(a, b, Z); id\rangle \overset{Red_4}{\rightsquigarrow}_{AS} \langle \sup\{v_1, v_2, v_3\}; \{Z/c\}\rangle$*, computing the fuzzy computed answer* $\langle \sup\{v_1, v_2, v_3\}; \{Z/c\}\rangle$ *but not the correct answer* $\langle \sup\{v_1, v_2\}; \{Z/Z_1\}\rangle$*.*

It is easy to give similar arguments justifying the need of reductants Red2 and Red3, by studying admissible computations for the program $\mathcal{P}$ *and the goals* $p(a, Y, c)$*) and* $p(X, b, c)$*, respectively.*

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:*

- $\langle p(a, b, c); id\rangle \overset{\mathcal{R}_1}{\rightsquigarrow}_{AS} \langle v_1; \{Y_1/b, Z_1/c\}\rangle$ *with f.c.a., fca4,* $\langle v_1; id\rangle$*,*
- $\langle p(a, b, c); id\rangle \overset{\mathcal{R}_2}{\rightsquigarrow}_{AS} \langle v_2; \{X_1/a, Z_1/c\}\rangle$ *with f.c.a., fca5,* $\langle v_2; id\rangle$*,*
- $\langle p(a, b, c); id\rangle \overset{\mathcal{R}_3}{\rightsquigarrow}_{AS} \langle v_3; \{X_1/a, Y_1/b\}\rangle$ *with f.c.a., fca6,* $\langle v_3; id\rangle$*.*

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

The previous example shows that all the 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. Moreover, 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". Hence, we have

to compute all the (critical) reductants associated with a program and not only the maximal critical reductants (that is, the improved G-reductants) if we want to maintain the (approximate) completeness. Therefore, as we are arguing, it is not an option to use just G-reductants, if we only compute the G-reductants, in general, the completeness result is lost.

6. Computing Reductants

We have just seen that a multi-adjoint program should contain 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*, outputs 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 6.1. *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 depicted using a Hasse diagram. Also, starting from the null configuration, we can organize the configurations by levels in such a way that if $\langle C, \theta\rangle$ is a configuration in level N, then the cardinality $|C| = N$. It is worth to remark that the configurations in level N can be obtained by joining all pairs of configurations in level $N - 1$.

Our algorithm takes the set of initial configurations as the starting point 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 $N-1$). Subsequently, the subsets of the new level are tested for unifiability. 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 configurations 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 $\theta_{ij} = mgu(C_i \cup C_j) \neq$ 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 taking each configuration in UnifConfs and applying Definition 4.2.

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 and the singleton elements (initial configurations). If we order the elements of S by inclusion, any level l of the corresponding Hasse 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

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

ARREGLAR ESTE PÁRRAFO (DEMASIADO INFORMAL) 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 are 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 it is common to use a pattern matching programming technique, what usually leads to rules whose heads do not unify. To the best of our knowledge, no algorithm has been previously developed to compute (all) the reductants of a program and, hence, it is not possible to make a thorough comparison between any other algorithm for computing reductants and the one introduced in this paper.

Summarizing, we have defined an algorithm for computing reductants that, from the very beginning, detects and discards configurations which are not unifiable. This way, the algorithm limits the number of unification steps which are needed to build reductants, gaining efficiency as much as possible.

7. Conclusions

We have revisited the notion of reductant in the framework of multi-adjoint logic programming. After surveying 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 some of the formal properties of the new notion of reductant and 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 property does not hold in general, but under very restrictive conditions. We have proved that any G-reductant can be transformed into a maximal critical reductant by using unfolding transformations. We have also shown, by means of a small but significative example, that it is necessary to compute all the reductants associated with a multi-adjoint logic program (and not only a significative subset of them: e.g., its maximal critical reductants). These reductants must be attached to the program in order to preserve the approximate completeness property of the multi-adjoint logic programming framework. Furthermore, 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 of [3] will also be studied.

Acknowledgements

The authors have been partially supported, respectively, by the Spanish MICINN projects TIN2013-45732-C4-2-P, TIN2012-39353-C04-01, and TIN2012-39353-C04-04.

References

[1] J. F. Baldwin, T. P. Martin, and B. W. Pilsworth. *Fril-Fuzzy and Evidential Reasoning in Artificial Intelligence*. John Wiley & Sons, 1995.

[2] T. H. Cao and N. V. Noi. A framework for linguistic logic programming. *Intl Journal of Intelligent Systems* 25(6):559–580, 2010.

[3] C. V. Damásio, J. Medina, and M. Ojeda-Aciego. A tabulation proof procedure for first-order residuated logic programs: Soundness, com-

pleteness and optimizations. In *Proc. of the Intl Conf on Fuzzy Systems, FUZZ-IEEE*, pages 2004–2011, 2006.

[4] P. Eklund, M. Galán, R. Helgesson, J. Kortelainen, G. Moreno, and C. Vázquez. Towards categorical fuzzy logic programming. *Lect Notes in Computer Science* 8256:109–121, 2013.

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

[6] 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.

[7] P. Hájek. *Metamathematics of fuzzy logic*. Springer, 1998.

[8] M. Ishizuka and N. Kanai. Prolog-ELF Incorporating Fuzzy Logic. *Proc. of the 9th Intl Joint Conf on Artificial Intelligence (IJCAI'85)*, pages 701–703. Morgan Kaufmann, 1985.

[9] N. Jones, C. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice-Hall, 1993.

[10] P. Julián-Iranzo, J. Medina, and M. Ojeda-Aciego. Revisiting reductants in the multi-adjoint logic programming framework. *Lecture Notes in Artificial Intelligence* 8761:694–702, 2014.

[11] P. Julián, G. Moreno, and J. Penabad. On Fuzzy Unfolding. A Multi-adjoint Approach. *Fuzzy Sets and Systems* 154:16–33, 2005.

[12] P. Julián, G. Moreno, and J. Penabad. An Improved Reductant Calculus using Fuzzy Partial Evaluation Techniques. *Fuzzy Sets and Systems* 160:162–181, 2009.

[13] M. Kifer and V. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming* 12:335–367, 1992.

[14] S. Krajči, R. Lencses, and P. Vojtáš. A comparison of fuzzy and annotated logic programming. *Fuzzy Sets and Systems* 144(1):173–192, 2004.

[15] T. Kuhr and V. Vychodil. Fuzzy logic programming reduced to reasoning with attribute implications. *Fuzzy Sets and Systems* 262:1–20, 2015.

[16] 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.

[17] J. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987. Second edition.

[18] J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Similarity-based unification: a multi-adjoint approach. *Fuzzy Sets and Systems* 146(1):43–62, 2004.

[19] P. Morcillo and G. Moreno. A practical approach for ensuring completeness of multi-adjoint logic computations via general reductants. In *Proc. of IX Jornadas sobre Programación y Lenguajes, PROLE'09, San Sebastián, Spain*, pages 355–363. 2009. (ISBN 978-84-692-4600-9).

[20] 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, Paris, France*, pages 138–145. IEEE, 2011.

[21] P. Morcillo and G. Moreno. Simplifying general reductants with unfolding-based techniques. In *Proc. of XI Jornadas sobre Programación y Lenguajes, PROLE'11, A Coruña, Spain*, pages 154–168, 2011. (ISBN 978-84-9749-487-8)

[22] 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.

[23] A. Pettorossi and M. Proietti. Rules and Strategies for Transforming Functional and Logic Programs. *ACM Computing Surveys* 28(2):360–414, 1996.

[24] P. Vojtáš. Fuzzy Logic Programming. *Fuzzy Sets and Systems* 124(1):361–370, 2001.

[25] H. Yasui, Y. Hamada, and M. Mukaidono. Fuzzy prolog based on Lukasiewicz implication and bounded product. In *Proc. of Intl Conf on Fuzzy Systems FUZZ-IEEE*, pages 949–954, 1995.