On Fuzzy Unfolding: A Multi–adjoint Approach

Pascual Julián [a] Ginés Moreno [b] Jaime Penabad [c]

[a] *Department of Computer Science*
ESI, Univ. of Castilla–La Mancha
Paseo de la Universidad, 4;
13071 Ciudad Real, Spain

[b] *Department of Computer Science*
[c] *Department of Mathematics*
EPSA, Univ. of Castilla–La Mancha
Campus Universitario, s/n;
02071 Albacete, Spain

Abstract

In the context of (fuzzy) logic programs, 'unfolding' means to transform a program rule by replacing an atom call of the body (of that rule) by its definition. Unfolding is a semantics–preserving program transformation technique that is able to improve programs, generating more efficient code, since it anticipates computation steps. Unfolding is the basis for developing sophisticated and powerful programming tools, such as fold/unfold transformation systems or partial evaluators. In this paper we address the problem of extending the classical definition of the unfolding rule (for pure logic programs) to the setting of multi–adjoint logic programming, where a *fuzzy computed answer* is a pair $\langle$*truth degree*; *substitution*$\rangle$ computed by a fuzzy generalization of the *modus ponens* inference rule. Our main contributions can be summarized as follows:

- We proved the independence of the computation rule for multi–adjoint admissible computations.
- Moreover, we defined a fuzzy unfolding rule and we demonstrated its strong correctness properties, that is, original and unfolded programs compute the same fuzzy computed answers for a given goal.
- We also proved that unfolding transformations increase the efficiency of the residual programs, by reducing the length of fuzzy admissible derivations when solving goals.

Key words: Fuzzy inference systems, fuzzy logic programming, unfolding

⋆ This work has been partially supported by the EU (FEDER) and the Spanish MEC, under grant TIN2004–07943–C04–03.

Email addresses: Pascual.Julian@uclm.es (Pascual Julián),
Gines.Moreno@uclm.es (Ginés Moreno),
Jaime.Penabad@uclm.es (Jaime Penabad).

1 Introduction

Fuzzy Logic Programming amalgamates fuzzy logic [1] and pure logic programming [2], in order to provide these traditional languages with techniques or constructs to deal with uncertainty and approximated reasoning. There is no common method for introducing fuzzy concepts into logic programming. We have found two major, and rather different, approaches:

- The first approach, represented by languages as LIKELOG [3], replaces the syntactic unification mechanism of classical SLD–resolution by a fuzzy unification algorithm, based on similarity relations (over constants and predicates). The fuzzy unification algorithm provides an extended most general unifier as well as a numerical value, called *unification degree*. Intuitively, the unification degree represents the truth degree associated with the (query) computed instance. Programs written in this kind of languages consist, in essence, in a set of ordinary (Prolog) clauses jointly with a set of "similarity equations" which play an important role during the unification process.

- For the second approach, programs are fuzzy subsets of (clausal) formulas, where the *truth degree* of each clause is explicitly annotated. The work of computing and propagating truth degrees relies on an extension of the resolution principle, whereas the (syntactic) unification mechanism remains untouched. An example of this kind of languages is the one described in [4].

We are mainly interested in the second class of fuzzy logic languages. Recently, it has been appeared in [5] a theoretical model for fuzzy logic programming which deals with many valued implications. This line of work has concluded with the introduction of a multi–adjoint logic programming framework[6–8]. This framework allows the use of multiple implications and rather general connectives in the rules of a program. Finally, in [9] we find an extremely flexible scheme where, apart from introducing negation and dealing with interval–valued fuzzy sets [1], each clause on a given program may be interpreted with a different logic.

Program transformation is an optimization technique for computer programs that starting with an initial program $\mathcal{P}_0$ derives a sequence $\mathcal{P}_1, \ldots, \mathcal{P}_n$ of transformed programs by applying *elementary transformation rules*. The aim is that the final program $\mathcal{P}_n$ have the same meaning as $\mathcal{P}_0$, but with a more efficient behaviour with regard to some criterion. Program transformation can be seen as a methodology for software development, hence its importance.

Among the elementary transformation rules the so called *unfolding* rule has been widely studied. In essence, an unfolding rule is a program transformation operation which replaces a program rule by the set of rules obtained after appli-

cation of a *symbolic computation* step (in all its possible forms) on the body of the selected rule [10]. Depending on the concrete paradigm taken into account (functional [11], logic [12] or integrated functional–logic [13]) the considered computation step will be performed using —some variant of— its associated operational mechanism (rewriting, resolution or narrowing, respectively). The unfolding rule is able to produce, by itself (i.e., without being combined with any other kind of transformation), important optimizations on the original program code. Beyond this initial benefit, different unfolding formulations have shown their usefulness in the construction of advanced techniques for program synthesis, program analysis, debugging, compiling, learning, and so on. But, perhaps, the fields where unfolding has exhibited its best properties and powerful capabilities were partial evaluation and fold/unfold–based program transformation [14].

Although one of the main goals of a transformation technique is to obtain a better behaviour of the transformed program with respect to some efficiency criterion (for instance, execution time), from the theoretical point of view, the main goal is to achieve the semantic correctness of the transformation. A proper formulation of the unfolding rule must guarantee that it is meaning preserving (i.e., the unfolded program must reproduce as closely as possible the "observable" effects of the original program). For this purpose, several "applicability" conditions must be identified without drastically reducing the class of programs where the transformation could be performed in a safe way. Language syntax and operational semantics play an important role in the identification of such requirements, as we will see in our extension of the unfolding transformation (for pure logic programs) to a fuzzy context.

In this paper we have selected the framework of multi–adjoint logic programming [6–8] for the formalization of the unfolding transformation, since this is a extremely general framework where the unfolding transformation rule can be defined in a natural way. In this language, a *fuzzy computed answer* is a pair $\langle \textit{truth degree}; \textit{substitution} \rangle$ computed by a fuzzy generalization of the *modus ponens* inference rule. We have proved a result which is the fuzzy counterpart of the independence of the computation rule theorem of [2] for multi–adjoint logic programs.

We have defined a fuzzy unfolding rule for multi–adjoint logic programs and, we have studied its correctness properties together with the theoretical gains in efficiency produced on the residual code. The major technical result consists in proving the strong soundness and completeness for the new unfolding rule, namely that the fuzzy answers computed by fuzzy admissible computations in the initial and the final program coincide, whereas the last one runs faster.

The outline of this paper is as follows. In the next section and Section 3, we recall the most important features of multi–adjoint logic programming and we

present the operational semantics of the language. In Section 4, the independence of the computation rule is formalized. In Section 5, we define the fuzzy unfolding rule, whereas in Section 6 we prove its main theoretical/practical properties. Finally, we show our conclusions in Section 7.

2 Fuzzy Logic Programs and Declarative Semantics

In [15], among the variety of fuzzy logic programming languages in the literature, we selected the one described in [4] for first defining the concept of unfolding of fuzzy logic programs; since we considered it was specially well–suited for this purpose, mainly due to its simple (but powerful) syntax and operational semantics. However, starting with [5], a line of research works generalizes fuzzy logic languages of this class by allowing simultaneous use of several implications in the rules and more general connectives in their bodies. Therefore, it is important to analyse and adapt those previous concepts on fuzzy unfolding for the new languages. In this section we summarize the main features of the *multi–adjoint logic programing* framework presented in [6–8] [1] .

Let $\mathcal{L}$ be a first order language containing variables, function symbols, predicate symbols, constants, quantifiers, $\forall$ and $\exists$, and several (arbitrary) connectives to capture different interdependencies between predicates:

$\wedge_1, \wedge_2, \ldots, \wedge_k$ `(conjunctions)`	$\leftarrow_1, \leftarrow_2, \ldots, \leftarrow_m$ `(implications)`
$\vee_1, \vee_2, \ldots, \vee_l$ `(disjunctions)`	$@_1, @_2, \ldots, @_n$ `(aggregations)`

The implication $\leftarrow_i$ is the left-arrow version of $\rightarrow_i$. Although the connectives $\wedge_i$, $\vee_i$ and $@_i$ are binary operators, we usually generalize them as functions with an arbitrary number of arguments. So, we write [2] , $@_i(x_1, \ldots, x_n)$ instead of $@_i(x_1, @_i(x_2, \ldots, @_i(x_{n-1}, x_n)\ldots))$.

Multi–adjoint logic programing is interested in a subset of formulas of the first order language $\mathcal{L}$ described above. A *rule* is a formula $A \leftarrow \mathcal{B}$, where A is an atomic formula (usually called the *head*) and $\mathcal{B}$ is a formula built from atomic formulas $B_1, \ldots, B_n$ — $n \geq 0$ — and conjunctors, disjunctors and aggregators (which is called the *body*). Rules with an empty body are called *facts*. A *goal* is a body submitted as a query to the system. Variables in a rule are assumed governed by universal quantifiers.

[1] See these works for a more formal introduction.

[2] Since "we have several connectives one cannot expect associativity and commutativity between them and hence parentheses should be used" [5]. Therefore, this way of writing must be seen as a notational convention that we use for convenience reasons.

Formulas are interpreted on a multi–adjoint lattice. To make this paper self–contained we reproduce the concepts of adjoint pair and multi–adjoint lattice from [6] and [7], which are grouped in the following definition:

Definition 1 *Let $\langle L, \preceq\rangle$ be a complete lattice. A* multi–adjoint lattice *is a tuple $\langle L, \preceq, \leftarrow_1, \&_1, \ldots, \leftarrow_n, \&_n\rangle$ satisfying the following conditions:*

(1) $\langle L, \preceq\rangle$ is bounded (i.e., it has a bottom, $\bot$, and top, $\top$, elements);
(2) $\top \,\&_i\, v = v \,\&_i\, \top = v$ for all $v \in L$ and $i = 1, \ldots, n$;
(3) Each $(\leftarrow_i, \&_i)$, with $i = 1, \ldots, n$, is and adjoint pair*, that is:*
  (a) Operation $\&_i$ is increasing in both arguments;
  (b) Operation $\leftarrow_i$ is increasing in the first argument and decreasing in the second argument;
  (c) Adjoint property*: for any $x, y, z \in L$, we have that $x \preceq (y \leftarrow_i z)$ holds iff $(x \&_i z) \preceq y$ holds.*

In order to be concrete, when representing truth values in the examples, we shall select L as the set of real numbers in the interval $[0, 1]$. But in general, L does not need to be a totally ordered set, it is useful any set equipped with a partial order and organized as complete bounded lattice. It is noteworthy that, $\&_i$ is not part of the object language $\mathcal{L}$, although it must be part of an extended language introduced in order to define the operational mechanism of multi–adjoint logic languages (see latter, in Section 3).

In the context of multi–adjoint logic programming it is possible to concentrate on Hebrand interpretations, disregarding general interpretations, in order to define a declarative semantics [8]. Therefore, a *fuzzy interpretation*, $\mathcal{I}$, is a mapping from the Herbrand base, $B_{\mathcal{L}}$, into the multi–adjoint lattice of truth values L. The truth value of a ground atom $A \in B_{\mathcal{L}}$ is $\mathcal{I}(A)$. As usual, for a specific assignment ϑ from terms into elements of the Herbrand universe $U_{\mathcal{L}}$, the valuation of a formula in an interpretation is obtained by structural induction on the complexity of that formula:

$$\begin{aligned}
\mathcal{I}(p(t_1, \ldots, t_n))[\vartheta] &= \mathcal{I}(p(t_1\vartheta, \ldots, t_n\vartheta)),\\
\mathcal{I}(c(A_1, \ldots, A_n))[\vartheta] &= [\![c]\!](\mathcal{I}(A_1)[\vartheta], \ldots, \mathcal{I}(A_n)[\vartheta]),\\
\mathcal{I}(A \leftarrow \mathcal{B})[\vartheta] &= \mathcal{I}(A)[\vartheta] [\![\leftarrow]\!] \mathcal{I}(\mathcal{B})[\vartheta],\\
\mathcal{I}((\forall x)\mathcal{A})[\vartheta] &= inf\{\mathcal{I}(\mathcal{A})[\vartheta'] \mid \vartheta' \;\; x\text{–equivalent to } \;\vartheta\},
\end{aligned}$$

where p is a predicate symbol, c an arbitrary connective, A and A_i atomic formulas, $\mathcal{B}$ any body, $\mathcal{A}$ any formula and we denote the meaning function (truth value function) for a connective c by $[\![c]\!]$. When the assignment would not be relevant, we shall omit it during the valuation of a formula.

We describe real problems with uncertain knowledge by means of fuzzy theories that constitute fuzzy logic programs.

Definition 2 *A* fuzzy theory *is a partial mapping T applying formulas into elements (truth values) of a lattice L.*

A multi–adjoint program, *$\mathcal{P}$, is a fuzzy theory such that the domain* $\mathrm{dom}(\mathcal{P})$ *is a finite set of rules and L is a multi–adjoint lattice equipped with several implications.*

Informally speaking, a multi–adjoint logic program can be seen as a set of pairs $\langle \mathcal{R}; \alpha \rangle$, where $\mathcal{R}$ is a rule and $\alpha = \mathcal{P}(\mathcal{R})$ is a *truth degree* expressing the confidence which the user of the system has in the truth of the rule $\mathcal{R}$. Often, we'll write "$\mathcal{R}$ with $\alpha = \mathcal{P}(\mathcal{R})$" instead of $\langle \mathcal{C}; \mathcal{P}(\mathcal{R}) \rangle$. Truth degrees are axiomatically assigned (for instance) by an expert.

An interpretation $\mathcal{I}$ is a *model* of a fuzzy theory if for any rule $\mathcal{R} \in \mathrm{dom}(\mathcal{P})$, $\mathcal{I}(\mathcal{R}) \geq \mathcal{P}(\mathcal{R})$.

In [8], the declarative semantics of a multi–adjoint logic program is given in terms of a *least fuzzy Herbrand model*. Also, its characterization by a fix point semantics is presented.

3 Operational Semantics and Extended Fuzzy Logic Language

In this section, we formalize the concepts of fuzzy admissible computation step, fuzzy admissible derivation and fuzzy computer answer, with slight variations with regard to the definitions that appear in [8].

In order to formalize the following definitions we need an extended language obtained by adding to the alphabet of the object language $\mathcal{L}$: i) elements of the lattice (truth values) and ii) an adjoint conjunction $\&_i$ for each implication $\leftarrow_i$ that appears in the multi–adjoint logic program. We will denote the extended language by $\mathcal{L}^e$, and formulas in $\mathcal{L}^e$ will be named $\mathcal{L}^e$–formulas. Informally, a $\mathcal{L}^e$– program is a set of pairs $\langle \mathcal{L}^e$–rule, truth value$\rangle$.

The operational mechanism that we are going to define is a backwards reasoning procedure that, using a generalization of *modus ponens*, provides a lower bound of the truth value of a goal under any model of a program. More precisely, starting from an extended goal and applying *modus ponens* on a selected atom A of the goal and a rule $\langle A' \leftarrow_i \mathcal{B}, v \rangle$ of the program, if there is a substitution $\theta = mgu(\{A = A'\})$ [3], we substitute the atom A by the ex-

[3] Let $mgu(E)$ denote the *most general unifier* of an equation set E (see [16] for a

tended expression $(v\&_i\mathcal{B})\theta$. The process is repeated until an extended formula with no atoms is obtained. Then the truth degree of the extended formula can be obtained by interpretation. The adjoint property of the pair $(\leftarrow_i, \&_i)$ warranties that the truth degree is a lower bound. Therefore, in this context, a computation can be seen as a two phase procedure with a first operational and a latter interpretive phase. Note that in other previous works [4,5] both phases were mixed.

In the formalization of the fuzzy admissible computation, we write $\mathcal{C}[A]$, or more generally $\mathcal{C}[A_1, \ldots, A_n]$, to denote a $\mathcal{L}^e$–formula where A, or $A_1, \ldots, A_n$ respectively, are sub-expressions (usually atoms) which arbitrarily occur in the —possibly empty— context $\mathcal{C}[\ ]$. Moreover, expression $\mathcal{C}[A/A']$ (and its obvious generalization) means the replacement of A by A' in the context $\mathcal{C}[\ ]$.

Definition 3 *Let $\mathcal{Q}$ be a $\mathcal{L}^e$–goal and let σ be a substitution, a* state *is a pair $\langle \mathcal{Q}; \sigma \rangle$. Let $\mathcal{E}$ be the set of states. Given a $\mathcal{L}^e$–program $\mathcal{P}$, we define a* Fuzzy Admissible Computation *as a state transition system, whose transition relation $\rightarrow_{AS} \subseteq (\mathcal{E} \times \mathcal{E})$ is the smallest relation satisfying the following* admissible rules*:*

Rule 1.

$$\langle \mathcal{Q}[A]; \sigma \rangle \rightarrow_{AS} \langle (\mathcal{Q}[A/v\&_i\mathcal{B}])\theta; \sigma\theta \rangle \text{ if } \begin{cases} (1)\ A \text{ is the selected atom in } \mathcal{Q}, \\ (2)\ \theta = mgu(\{A' = A\}), \\ (3)\ \langle A' \leftarrow_i \mathcal{B}; v \rangle \text{ in } \mathcal{P} \text{ and } \mathcal{B} \text{ is not empty.} \end{cases}$$

Rule 2.

$$\langle \mathcal{Q}[A]; \sigma \rangle \rightarrow_{AS} \langle (\mathcal{Q}[A/v])\theta; \sigma\theta \rangle \text{ if } \begin{cases} (1)\ A \text{ is the selected atom in } \mathcal{Q}, \\ (2)\ \theta = mgu(\{A' = A\}), \text{ and} \\ (3)\ \langle A' \leftarrow; v \rangle \text{ in } \mathcal{P}. \end{cases}$$

Rule 3.

$$\langle \mathcal{Q}[A]; \sigma \rangle \rightarrow_{AS} \langle (\mathcal{Q}[A/\bot]); \sigma \rangle \text{ if } \begin{cases} (1)\ A \text{ is the selected atom in } \mathcal{Q}, \text{ and} \\ (2)\ \text{there is no rule in } \mathcal{P} \text{ whose head unifies with } A. \end{cases}$$

All familiar logic programming concepts (such as derivation, etc.) can be extended for the fuzzy case, assuming also that formulas involved in fuzzy admissible computation steps are renamed before being used. Note that, Rule 3 is

formal definition of this concept).

introduced to cope with (possible) unsuccessful admissible derivations. In the following, symbols $\rightarrow_{AS1}$, $\rightarrow_{AS2}$ and $\rightarrow_{AS3}$ may be used for explicitly referring to the application of each one of the admissible rules. When needed, the exact rule used in the corresponding step will be annotated as a super–index of the $\rightarrow_{AS}$ symbol. Also we shall use ${\rightarrow_{AS}}^n$ to denote a sequence of n admissible computation steps and ${\rightarrow_{AS}}^*$ for an arbitrary sequence of steps.

Definition 4 *Let $\mathcal{P}$ be a $\mathcal{L}^e$–program and $\mathcal{Q}$ be a $\mathcal{L}^e$–goal. A sequence $\mathcal{E}_0 \rightarrow_{AS} \mathcal{E}_1 \rightarrow_{AS}{}^* \mathcal{E}_n$ is a* successful *admissible derivation if:*

(1) $\mathcal{E}_0 = \langle \mathcal{Q}; id \rangle$, where id is the empty substitution;
(2) for each $0 \leq i < n$, $\mathcal{E}_i \rightarrow_{AS} \mathcal{E}_{i+1}$ is an admissible derivation step;
(3) $\mathcal{E}_n = \langle \mathcal{Q}'; \theta' \rangle$ and $\mathcal{Q}'$ is a $\mathcal{L}^e$–formula that does not contain atoms.

Note that, if after a sequence of admissible steps a $\mathcal{L}^e$–goal becomes a $\mathcal{L}^e$–formula that does not contain atoms, then it can be interpreted directly in the multi–adjoint lattice L. This justifies the following extension of the notion of computed answer to our fuzzy setting. In this definition we use $\mathcal{V}ar(s)$ to refer to the set of distinct variables occurring in the syntactic object s, and $\theta[\mathcal{V}ar(s)]$ denotes the substitution obtained from θ by restricting its domain, $Dom(\theta)$, to $\mathcal{V}ar(s)$.

Definition 5 *Let $\mathcal{P}$ be a $\mathcal{L}^e$–program and $\mathcal{Q}$ be a $\mathcal{L}^e$–goal. Given a successful admissible derivation $\langle \mathcal{Q}; id \rangle \rightarrow_{AS}{}^* \langle @(r_1, \ldots, r_n); \theta \rangle$, with $r_i \in L$ for any $i \in \{1, \ldots, n\}$, the pair $\langle [\![@]\!](r_1, \ldots, r_n); \sigma \rangle$, where $\sigma = \theta[\mathcal{V}ar(\mathcal{Q})]$, is a* fuzzy computed answer *(f.c.a.) for that derivation.*

We illustrate the previous concepts and the last definition by means of an example.

Example 6 *Let $\mathcal{P}$ be the following $\mathcal{L}$–program,*

$$\begin{array}{lll} \mathcal{R}_1: & p(X) \leftarrow_{\texttt{prod}} q(X,Y) \wedge_{\texttt{G}} r(Y) & \textit{with } \alpha = 0.8 \\ \mathcal{R}_2: & q(a,Y) \leftarrow_{\texttt{prod}} s(Y) & \textit{with } \alpha = 0.7 \\ \mathcal{R}_3: & q(Y,a) \leftarrow_{\texttt{luka}} r(Y) & \textit{with } \alpha = 0.8 \\ \mathcal{R}_4: & r(Y) \leftarrow & \textit{with } \alpha = 0.7 \\ \mathcal{R}_5: & s(b) \leftarrow & \textit{with } \alpha = 0.9 \end{array}$$

The labels `prod`, `G` *and* `luka` *mean for* product logic, Gödel intuitionistic logic *and* Łukasiewicz logic *respectively. That is, $[\![\&_{\texttt{prod}}]\!](x,y) = x \cdot y$, $[\![\wedge_{\texttt{G}}]\!](x,y) = min(x,y)$, and $[\![\&_{\texttt{luka}}]\!](x,y) = max(0, x+y-1)$.*

In the following successful admissible derivation for the program $\mathcal{P}$ and the goal $\leftarrow p(X) \wedge_{\texttt{G}} r(a)$, we underline the selected expression in each admissible step:

$$\langle \underline{p(X)} \wedge_{\mathtt{G}} r(a); id\rangle \rightarrow_{AS1}{}^{\mathcal{R}_1} \langle (0.8 \&_{\mathtt{prod}} (\underline{q(X_1, Y_1)} \wedge_{\mathtt{G}} r(Y_1))) \wedge_{\mathtt{G}} r(a); \sigma_1\rangle$$

$$\rightarrow_{AS1}{}^{\mathcal{R}_2} \langle (0.8 \&_{\mathtt{prod}} ((0.7 \&_{\mathtt{prod}} \underline{s(Y_2)}) \wedge_{\mathtt{G}} r(Y_2))) \wedge_{\mathtt{G}} r(a); \sigma_2\rangle$$

$$\rightarrow_{AS2}{}^{\mathcal{R}_5} \langle (0.8 \&_{\mathtt{prod}} ((0.7 \&_{\mathtt{prod}} 0.9) \wedge_{\mathtt{G}} \underline{r(b)})) \wedge_{\mathtt{G}} r(a); \sigma_3\rangle$$

$$\rightarrow_{AS2}{}^{\mathcal{R}_4} \langle (0.8 \&_{\mathtt{prod}} ((0.7 \&_{\mathtt{prod}} 0.9) \wedge_{\mathtt{G}} 0.7)) \wedge_{\mathtt{G}} \underline{r(a)}; \sigma_4\rangle$$

$$\rightarrow_{AS2}{}^{\mathcal{R}_4} \langle (0.8 \&_{\mathtt{prod}} ((0.7 \&_{\mathtt{prod}} 0.9) \wedge_{\mathtt{G}} 0.7)) \wedge_{\mathtt{G}} 0.7; \sigma_5\rangle$$

where $\sigma_1 = \{X/X_1\}$, $\sigma_2 = \{X/a, X_1/a,\ Y_1/Y_2\}$, $\sigma_3 = \{X/a, X_1/a, Y_1/b,\ Y_2/b\}$, $\sigma_4 = \{X/a, X_1/a,\ Y_1/b, Y_2/b, Y_3/b\}$ *and* $\sigma_5 = \{X/a, X_1/a,\ Y_1/b, Y_2/b,\ Y_3/b, Y_4/a\}$.

Then the f.c.a for this admissible derivation is $\langle 0.504, \{X/a\}\rangle$, *since:*

$$\mathcal{I}((0.8 \&_{\mathtt{prod}} ((0.7 \&_{\mathtt{prod}} 0.9) \wedge_{\mathtt{G}} 0.7)) \wedge_{\mathtt{G}} 0.7) = \cdots$$

$$= (0.8 [\![\&_{\mathtt{prod}}]\!] ((0.7 [\![\&_{\mathtt{prod}}]\!] 0.9) [\![\wedge_{\mathtt{G}}]\!] 0.7)) [\![\wedge_{\mathtt{G}}]\!] 0.7 = \cdots = 0.504 [\![\wedge_{\mathtt{G}}]\!] 0.7 = 0.504$$

and $\sigma_5[\mathcal{V}ar(\mathcal{Q})] = \{X/a\}$.

In an adjoint pair, the intended meaning for $\leftarrow_i$ is an implication, coupled joinly with a conjunction operator $\&_i$ evaluating *modus ponens* in such a way that it is a sound inference rule:

$$\frac{\langle (A \leftarrow_i B), x\rangle \qquad \langle B, y\rangle}{\langle A, \&_i(x, y)\rangle}$$

That is, if an interpretation $\mathcal{I}$ is a model of $A \leftarrow_i B$ (i.e., $\mathcal{I}(A \leftarrow_i B) \succeq x$) and B (i.e., $\mathcal{I}(B) \succeq y$) then $\mathcal{I}$ is a model of A (i.e., $\mathcal{I}(A) \succeq x \&_i y$). The soundness of the operational mechanism can be stated as follows: $x \preceq \mathcal{I}(A \leftarrow_i B) = \mathcal{I}(A) [\![\leftarrow_i]\!] \mathcal{I}(B) \preceq \mathcal{I}(A) [\![\leftarrow_i]\!] y$, since "$\leftarrow_i$" is decreasing in its second argument and $\mathcal{I}(B) \succeq y$; then, by the adjoint property, $\mathcal{I}(A) \succeq x \&_i y$.

In [8], the authors established an *approximative* completeness result for the operational mechanism of multi–adjoint logic programming. However, the completeness result can be achieve by restricting the properties of connectives [5].

As for the classical SLD–Resolution calculus, we assume the existence of a fixed selection function, also called *fuzzy computation rule*, deciding, for a given goal, which is the selected expression to be exploited in the next fuzzy admissible step. For instance, when building the admissible derivation shown in Example 6, we have used a computation rule similar to the left to right

selection rule of Prolog. Given a fuzzy computation rule $\mathcal{S}$, we say that a fuzzy admissible derivation is *via* $\mathcal{S}$ if the selected expression in every admissible step is obtained by the application of the mapping $\mathcal{S}$ to the corresponding goal in that step. In the following section, we establish in our fuzzy setting the independence of the computation rule proved in [2] for the pure logic programming case.

4 Independence of the Fuzzy Computation Rule

Before starting with the core of the paper, we introduce some technical notations and concepts that will help us to develop our proofs. In the following, we use $\mathcal{R} \ll \mathcal{P}$ to denote an *standardised apart* new variant [4] of a program rule such that $\mathcal{R}$ contains no variable which was previously met during a computation. The equational representation of a substitution $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ is the set of equations $\widehat{\theta} = \{x_1 = t_1, \ldots, x_n = t_n\}$. *Parallel composition* of substitutions [17] corresponds to the notion of unification generalized to substitutions. Given two idempotent substitutions θ_1 and θ_2, the parallel composition $\theta_1 \Uparrow \theta_2 = mgu(\widehat{\theta_1} \cup \widehat{\theta_2})$. The following property will be useful later.

Proposition 7 *[17] Let θ_1 and θ_2 be idempotent substitutions. Then,*

$$\theta_1 \Uparrow \theta_2 = \theta_1 mgu(\widehat{\theta_2}\theta_1) = \theta_2 mgu(\widehat{\theta_1}\theta_2).$$

In order to prove the independence of the fuzzy computation rule, we need the following auxiliary Lemmas. The first one focuses on the preservation of substitutions in f.c.a.'s obtained by the application of two admissible steps done with the first and/or the second rules of Definition 3, with independence of the order in which atoms are exploited.

Lemma 8 *Let $\mathcal{P}$ be a $\mathcal{L}^e$–program and let $\mathcal{Q}_0[A, A']$ be a $\mathcal{L}^e$–goal. If in the following derivations we only consider steps of kinds $\rightarrow_{AS1}$ and $\rightarrow_{AS2}$, then,*

$\langle \mathcal{Q}_0; \theta_0 \rangle \rightarrow_{AS}{}^{A} \langle \mathcal{Q}_1; \theta_0\theta_1 \rangle \rightarrow_{AS}{}^{A'\theta_1} \langle \mathcal{Q}_2; \theta_0\theta_1\theta_2 \rangle$ iff
$\langle \mathcal{Q}_0; \theta_0 \rangle \rightarrow_{AS}{}^{A'} \langle \mathcal{Q}'_1; \theta_0\theta'_1 \rangle \rightarrow_{AS}{}^{A\theta'_1} \langle \mathcal{Q}'_2; \theta_0\theta'_1\theta'_2 \rangle$, where $\theta_0\theta_1\theta_2 = \theta_0\theta'_1\theta'_2$.

PROOF. ($\Rightarrow$) Let atoms H_1 and H_2 be the heads of rules $\mathcal{R}_1, \mathcal{R}_2 \ll \mathcal{P}$ used to exploit (instances of) atoms A and A', respectively, in the considered derivation: $\langle \mathcal{Q}_0; \theta_0 \rangle \rightarrow_{AS}{}^{A} \langle \mathcal{Q}_1; \theta_0\theta_1 \rangle \rightarrow_{AS}{}^{A'\theta_1} \langle \mathcal{Q}_2; \theta_0\theta_1\theta_2 \rangle$, where we have that $\theta_1 = mgu(\{A = H_1\})$ and $\theta_2 = mgu(\{A'\theta_1 = H_2\})$. Moreover, since we don't consider steps of kind $\rightarrow_{AS3}$, then $\theta_0\theta_1\theta_2 \neq fail$, and in particular $\theta_2 \neq fail$, which also implies that $\theta'_1 = mgu(\{A' = H_2\}) \neq fail$. Now, the following equalities hold:

[4] Formally, we say that two expressions E_1 and E_2 are variants one of each other if and only if there are some renaming substitutions ρ and ρ' such that $E_1 = E_2\rho$ and $E_2 = E_1\rho'$.

$$
\begin{array}{ll}
\theta_1\theta_2 & = \\
\theta_1 mgu(\{A'\theta_1 = H_2\}) & = (\text{since } Dom(\theta_1) \cap \mathcal{V}ar(\mathcal{R}_2) = \emptyset) \\
\theta_1 mgu(\widehat{mgu}(\{A' = H_2\})\theta_1) & = \\
\theta_1 mgu(\widehat{\theta'_1}\theta_1) & = (\text{by Proposition 7}) \\
\theta_1 \Uparrow \theta'_1 & = (\text{by Proposition 7}) \\
\theta'_1 mgu(\widehat{\theta_1}\theta'_1) & = \\
\theta'_1 mgu(\widehat{mgu}(\{A = H_1\})\theta'_1) & = (\text{since } Dom(\theta'_1) \cap \mathcal{V}ar(\mathcal{R}_1) = \emptyset) \\
\theta'_1 mgu(\{A\theta'_1 = H_1\}) &
\end{array}
$$

Moreover, since $\theta_1\theta_2 \neq fail$ then $\theta'_1 mgu(\{A\theta'_1 = H_1\}) \neq fail$ and, in particular, $\theta'_2 = mgu(\{A\theta'_1 = H_1\}) \neq fail$. Hence, $\theta_1\theta_2 = \theta'_1\theta'_2$ which implies that $\theta_0\theta_1\theta_2 = \theta_0\theta'_1\theta'_2$, as we wanted to prove.

($\Leftarrow$) This case can be easily proved in a similar way as the previous one, by also exploiting the equivalence between $\theta_1\theta_2$ and $\theta'_1\theta'_2$.

The following Lemma generalizes Lemma 8 at two different levels:

- first, preserving both elements in derivation states, i.e., not only substitution, but also partially evaluated truth degrees; and
- second, using the whole set of admissible rules of Definition 3 (instead of the first pair only) when applying the two admissible steps on the considered derivations.

Lemma 9 (Switching Lemma) *Let $\mathcal{P}$ be a $\mathcal{L}^e$–program and let $\mathcal{Q}_0[A, A']$ be a $\mathcal{L}^e$–goal. Then, $\langle \mathcal{Q}_0; \theta_0 \rangle \rightarrow_{AS}{}^{A} \langle \mathcal{Q}_1; \theta_1 \rangle \rightarrow_{AS}{}^{A'\theta_1} \langle \mathcal{Q}_2; \theta_2 \rangle$ iff $\langle \mathcal{Q}_0; \theta_0 \rangle \rightarrow_{AS}{}^{A'} \langle \mathcal{Q}'_1; \theta'_1 \rangle \rightarrow_{AS}{}^{A\theta'_1} \langle \mathcal{Q}'_2; \theta'_2 \rangle$, where $\mathcal{Q}_2 = \mathcal{Q}'_2$ and $\theta_2 = \theta'_2$.*

PROOF.

For readability reasons, we underline the selected atoms exploited in each derivation step. In our proof we exhaustively proceed with each one of all the possible cases. Fortunately, note that the case where the first step is done with rule i and the second one with rule j is perfectly analogous to the case where the first step is done with rule j and the second one with rule i, which drastically reduces the number of alternatives.

(1) First step with Rule 1 and second step with Rule 1.
Assume that $\mathcal{R}_1 : \langle H_1 \leftarrow_1 \mathcal{B}_1; v_1 \rangle \ll \mathcal{P}$ and $\mathcal{R}_2 : \langle H_2 \leftarrow_2 \mathcal{B}_2; v_2 \rangle \ll \mathcal{P}$. Then,

$$\langle \mathcal{Q}_0[\underline{A}, A']; \theta_0 \rangle \quad \rightarrow_{AS1}{}^{\mathcal{R}_1}$$
$$\langle \mathcal{Q}_0[A/(v_1 \&_1 \mathcal{B}_1), \underline{A'}]\theta_1; \theta_1 \rangle \quad \rightarrow_{AS1}{}^{\mathcal{R}_2}$$
$$\langle \mathcal{Q}_0[A/(v_1 \&_1 \mathcal{B}_1), A'/(v_2 \&_2 \mathcal{B}_2)]\theta_2; \theta_2 \rangle$$

iff

$$\langle \mathcal{Q}_0[A, \underline{A'}]; \theta_0 \rangle \quad \rightarrow_{AS1}{}^{\mathcal{R}_2}$$
$$\langle \mathcal{Q}_0[\underline{A}, A'/(v_2 \&_2 \mathcal{B}_2)]\theta'_1; \theta'_1 \rangle \quad \rightarrow_{AS1}{}^{\mathcal{R}_1}$$
$$\langle \mathcal{Q}_0[A/(v_1 \&_1 \mathcal{B}_1), A'/(v_2 \&_2 \mathcal{B}_2)]\theta'_2; \theta'_2 \rangle$$

By Lemma 8, $\theta_2 = \theta'_2$, which also implies that the first element of the final states in both derivations are syntactically identical, as we wanted to prove.

(2) First step with Rule 1 and second step with Rule 2.
Now we assume that $\mathcal{R}_1 : \langle H_1 \leftarrow_1 \mathcal{B}_1; v_1 \rangle \ll \mathcal{P}$ and $\mathcal{R}_2 : \langle H_2 \leftarrow; v_2 \rangle \ll \mathcal{P}$. The reader may easily check that this case is perfectly analogous to the previous one, but now, instead of using $\rightarrow_{AS1}{}^{\mathcal{R}_2}$ we need $\rightarrow_{AS2}{}^{\mathcal{R}_2}$, whereas expression $A'/(v_2 \&_2 \mathcal{B}_2)$ can be directly simplified to A'/v_2.

(3) First step with Rule 1 and second step with Rule 3.
In this case we assume that atom A unifies with the head of a program rule $\mathcal{R}_1 : \langle H_1 \leftarrow_1 \mathcal{B}_1; v_1 \rangle \ll \mathcal{P}$, whereas there is no rule in the program whose head unify with (any instance of) atom A'. It is important to note that, in contrast with the previous cases, only one unifier (instead of two ones) is computed in the considered derivations. Then,

$$\langle \mathcal{Q}_0[\underline{A}, A']; \theta_0 \rangle \quad \rightarrow_{AS1}{}^{\mathcal{R}_1}$$
$$\langle \mathcal{Q}_0[A/(v_1 \&_1 \mathcal{B}_1), \underline{A'}]\theta_1; \theta_1 \rangle \quad \rightarrow_{AS3}$$
$$\langle \mathcal{Q}_0[A/(v_1 \&_1 \mathcal{B}_1), A'/\bot]\theta_1; \theta_1 \rangle$$

iff

$$\langle \mathcal{Q}_0[A, \underline{A'}]; \theta_0 \rangle \quad \rightarrow_{AS3}$$
$$\langle \mathcal{Q}_0[\underline{A}, A'/\bot]; \theta_0 \rangle \quad \rightarrow_{AS1}{}^{\mathcal{R}_1}$$
$$\langle \mathcal{Q}_0[A/(v_1 \&_1 \mathcal{B}_1), A'/\bot]\theta_1; \theta_1 \rangle$$

as we wanted to prove.

(4) First step with Rule 2 and second step with Rule 2.
This case is perfectly analogous to the first one, but now all the admissible steps are of type $\rightarrow_{AS2}$ instead of $\rightarrow_{AS1}$, and all the occurrences of

expressions $A/(v_1 \&_1 \mathcal{B}_1)$ and $A'/(v_2 \&_2 \mathcal{B}_2)$ in the considered goals, can be directly simplified to A/v_1 and A'/v_2, respectively.

(5) First step with Rule 2 and second step with Rule 3.
We omit the proof of this case once again, since it can be easily obtained by exploiting the similarities between the present situation and the third one.

(6) First step with Rule 3 and second step with Rule 3.
In our final proof, we don't need to compute unifiers (which strongly contrast with any other case), thus obtaining: $\langle \mathcal{Q}_0[\underline{A}, A']; \theta_0 \rangle \rightarrow_{AS3} \langle \mathcal{Q}_0[A/\bot, \underline{A'}]; \theta_0 \rangle \rightarrow_{AS3} \langle \mathcal{Q}_0[A/\bot, A'/\bot]; \theta_0 \rangle$ iff $\langle \mathcal{Q}_0[A, \underline{A'}]; \theta_0 \rangle \rightarrow_{AS3} \langle \mathcal{Q}_0[\underline{A}, A'/\bot]; \theta_0 \rangle \rightarrow_{AS3} \langle \mathcal{Q}_0[A/\bot, A'/\bot]; \theta_0 \rangle$, as we wanted to prove.

Now, we can formalize and prove the main result of this section.

Theorem 10 (Independence of the Fuzzy Computation Rule) *Let $\mathcal{P}$ be a $\mathcal{L}^e$–program and let $\mathcal{Q}$ be a $\mathcal{L}^e$–goal. For any pair of fuzzy computation rules $\mathcal{S}$ and $\mathcal{S}'$, we have that: $\langle \mathcal{Q}; id \rangle \rightarrow_{AS}{}^{k}_{\mathcal{S}} \langle @(r_1, \ldots, r_n); \theta \rangle$ iff $\langle \mathcal{Q}; id \rangle \rightarrow_{AS}{}^{k}_{\mathcal{S}'} \langle @(r_1, \ldots, r_n); \theta \rangle$ where $r_i \in L$ for any $i \in \{1, \ldots, n\}$ and k is the (same) number of fuzzy admissible steps in both derivations.*

PROOF. Immediate by repeatedly applying the Switching Lemma 9.

5 Fuzzy Unfolding of Extended Programs

As we have seen in the previous sections, the differences between $\mathcal{L}$–programs and $\mathcal{L}^e$–programs appear only at the syntactic level: the bodies of $\mathcal{L}^e$–rules (which intuitively have the same structure of any initial, intermediate or final goal appearing in fuzzy admissible derivations) are similar to the bodies of $\mathcal{L}$–rules (in essence, simple original goals) but possibly including elements of the lattice (truth values) and adjoint conjunctions of the form $\&_i$. This implies that any $\mathcal{L}$–program is also an $\mathcal{L}^e$–program, although the contrary is not always true (i.e., the set of $\mathcal{L}$–programs is a proper subclass of the set of $\mathcal{L}^e$–programs). Apart from this simple fact (which, on the other hand, is mandatory to define the notion of fuzzy admissible computation) both languages share all kind of semantics, like obviously the operational one, but also the declarative and the least fix–point ones and their correctness/ completeness properties, as described in [8].

On the other hand, as we know, the unfolding rule consists in essence in the application of a symbolic computation step on (the selected atom of) the body

of a rule which, in our fuzzy setting, corresponds to the application of any of the three rules described in Definition 3. Observe that this process always generates rules whose bodies include truth degrees and possibly adjoint conjunctions. Hence, an unfolding transformation based on fuzzy admissible computations is able to preserve the syntactic structure of $\mathcal{L}^e$–programs but, even in the case that original programs be also $\mathcal{L}$–programs, the transformed ones will never belong to this subclass: the truth degrees and adjoint conjunctions incorporated on the body of transformed rules by unfolding steps force the lost of the original $\mathcal{L}$–program syntax. In order to avoid this inconvenience, our following definition focuses in the general framework of $\mathcal{L}^e$–programs instead of the more restricted subclass of $\mathcal{L}$–programs.

Definition 11 *Let $\mathcal{P}$ be an $\mathcal{L}^e$–program and $\mathcal{R} : (A \leftarrow \mathcal{B}$ with $\alpha = v) \in \mathcal{P}$ a (non unit) $\mathcal{L}^e$–program rule. Then, the fuzzy unfolding of program $\mathcal{P}$ with respect to rule $\mathcal{R}$ is the new $\mathcal{L}^e$–program $\mathcal{P}' = (\mathcal{P} - \{\mathcal{R}\}) \cup \mathcal{U}$ such that:*

$$\mathcal{U} \;=\; \{A\sigma \leftarrow \mathcal{B}' \text{ with } \alpha = v \;\mid\; \langle \mathcal{B}; id\rangle \rightarrow_{AS} \langle \mathcal{B}'; \sigma\rangle\}$$

There are some remarks to do regarding our definition. Similarly to the classical SLD–resolution based unfolding rule presented in [12], the substitutions computed by admissible steps during unfolding are incorporated to the transformed rules in a natural way, i.e., by applying them to the head of the rule. On the other hand, regarding the propagation of truth degrees, we solve this problem in a very easy way: the unfolded rule directly inherits the truth degree α of the original rule.

However, a deeper analysis of the unfolding transformation reveals us that the body of the transformed rule also contains 'compiled–in' information on both components of a fuzzy computed answer (i.e., truth degree and substitution). Regarding truth degrees, we observe that the body of the transformed rule may include: i) the symbol $\bot$, provided a $\rightarrow_{AS3}$ admissible step is performed; ii) a truth degree, coming from the second rule involved in the unfolding step, when the admissible step is done with $\rightarrow_{AS2}$; or iii) a truth degree together with the adjoint conjunction associated to the second rule involved in the unfolding step, if the admissible step is based on $\rightarrow_{AS1}$. Summarizing, the propagation of truth degrees during unfolding is done at two different levels:

(1) by directly assigning the truth degree of the original rule as the truth degree of the transformed one, and
(2) by introducing new truth degrees (of other rules or alternatively $\bot$) and possibly adjoint conjunctions in its body.

Those manipulations in the body of the rule will drastically affect the computation/propagation of truth degrees when solving goals against transformed programs. Let us now illustrate all these facts with an example.

Example 12 *Consider again program $\mathcal{P}$ shown in Example 6. It is easy to see*

that the unfolding of program $\mathcal{P}$ w.r.t. rule $\mathcal{R}_2$ (exploiting the second admissible rule of Definition 3) generates the new program $\mathcal{P}' = (\mathcal{P} - \{\mathcal{R}_2\}) \cup \{\mathcal{R}_{25}\}$, where $\mathcal{R}_{25}$ is the new unfolded rule $q(a,b) \leftarrow_{\mathtt{prod}} 0.9$ with $\alpha = 0.7$. On the other hand, if we want to unfold now rule $\mathcal{R}_1$ in program $\mathcal{P}'$, we must firstly build the following one–step admissible derivations[5] *:*

$$\langle \underline{q(X,Y)} \wedge_{\mathtt{G}} r(Y); id \rangle \rightarrow_{AS1}{}^{\mathcal{R}_{25}} \langle (0.7 \&_{\mathtt{prod}} 0.9) \wedge_{\mathtt{G}} r(b); \{X/a, Y/b\} \rangle$$

$$\langle \underline{q(X,Y)} \wedge_{\mathtt{G}} r(Y); id \rangle \rightarrow_{AS1}{}^{\mathcal{R}_{3}} \langle (0.8 \&_{\mathtt{luka}} r(Y_1)) \wedge_{\mathtt{G}} r(a); \{X/Y_1, Y/a\} \rangle$$

So, the unfolded program $\mathcal{P}'' = (\mathcal{P}' - \{\mathcal{R}_1\}) \cup \{\mathcal{R}_{125}, \mathcal{R}_{13}\}$, where:

$$\mathcal{R}_{125} : p(a) \leftarrow_{\mathtt{prod}} (0.7 \&_{\mathtt{prod}} 0.9) \wedge_{\mathtt{G}} r(b) \qquad \text{with } \alpha = 0.8$$

$$\mathcal{R}_{13} : \; p(Y_1) \leftarrow_{\mathtt{prod}} (0.8 \&_{\mathtt{luka}} r(Y_1)) \wedge_{\mathtt{G}} r(a) \quad \text{with } \alpha = 0.8$$

Moreover, by performing a new admissible step with the second rule of Definition 3 on the body of rule $\mathcal{R}_{125}$, we obtain the new unfolded rule $p(a) \leftarrow_{\mathtt{prod}} (0.7 \&_{\mathtt{prod}} 0.9) \wedge_{\mathtt{G}} 0.7$ with $\alpha = 0.8$. It is important to note that the application of this last rule to the goal proposed in Example 6 simulates the effects of the first four admissible steps shown in the derivation of the same example, which evidences the improvements achieved by unfolding on transformed programs.

The most important and practical purpose of the unfolding transformation, apart from preserving the program semantics, is to optimize code, independently of the object language. Classical fold/unfold based transformation systems optimize programs by returning code which uses the same source language, but unfolding has also played important roles in the design of compilers (see [18]) that generate an object code written in a target language. In this sense, our unfolding transformation can be seen as a mixed technique that optimizes $\mathcal{L}$–programs and compiles it into $\mathcal{L}^e$–programs, with the advantage in our case that both programs are executable with exactly the same operational principle, apart from sharing any other kind of semantics and related properties.

The following section is devoted to establish the best properties one can expect of a transformation like our fuzzy unfolding, namely:

- on the theoretical side, the exact and total correspondence between fuzzy computed answers for goals executed against the original and the transformed programs, and
- on the practical side, the gains in efficiency of unfolded programs that is possible to obtain by reducing the number of fuzzy admissible steps needed to solve a goal.

[5] Note that unfolding steps are always performed using a fixed computation rule. In this and the other examples we are using a left-to-right Prolog like computation rule.

6 Properties of Fuzzy Unfolding

We start this section by proving the following Lemma, which can be seen as the counterpart of Lemma 8 (preservation of substitutions in f.c.a.'s on interchangeable derivation steps). Intuitively it shows that, even in the case that two admissible steps can not be switched, since the second step exploits an atom introduced on the considered goal by the first one, their effect (w.r.t. fuzzy computed answer substitutions) can be simulated by a single step performed using a transformed rule obtained by fuzzy unfolding.

Lemma 13 *Let $\mathcal{P}$ be a $\mathcal{L}^e$–program, $\mathcal{Q}_0$ a $\mathcal{L}^e$–goal and $\mathcal{R}_1, \mathcal{R}_2 \ll \mathcal{P}$. Then, $\langle \mathcal{Q}_0; \theta_0 \rangle \rightarrow_{AS1}{}^{\mathcal{R}_1} \langle \mathcal{Q}_1; \theta_0\theta_1 \rangle \rightarrow_{AS}{}^{\mathcal{R}_2} \langle \mathcal{Q}_2; \theta_0\theta_1\theta_2 \rangle$, where the second step is of kind $\rightarrow_{AS1}$ or $\rightarrow_{AS2}$ and exploits an atom introduced in $\mathcal{Q}_1$ after the first step, iff $\langle \mathcal{Q}_0; \theta_0 \rangle \rightarrow_{AS1}{}^{\mathcal{R}_3} \langle \mathcal{Q}_3; \theta_0\theta_3 \rangle$, where $\mathcal{R}_3$ is obtained by fuzzy unfolding of $\mathcal{R}_1$ using $\mathcal{R}_2$, such that $\theta_0\theta_1\theta_2 = \theta_0\theta_3$ $[\mathcal{V}ar(\mathcal{Q}_0)]$.*

PROOF.
($\Rightarrow$) Let $\mathcal{R}_1 : \langle H_1 \leftarrow_1 \mathcal{B}_1[A_1]; v_1 \rangle$, let H_2 be the atom at the head of rule $\mathcal{R}_2$ and assume that A is the atom selected in $\mathcal{Q}_0$ by the considered computation rule. Then, in the admissible derivation $\langle \mathcal{Q}_0;\theta_0 \rangle \rightarrow_{AS1}{}^{\mathcal{R}_1} \langle \mathcal{Q}_1;\theta_0\theta_1 \rangle \rightarrow_{AS}{}^{\mathcal{R}_2} \langle \mathcal{Q}_2;\theta_0\theta_1\theta_2 \rangle$ we have that: $\theta_1 = mgu(\{A = H_1\})$ and $\mathcal{Q}_1 = \mathcal{Q}_0[A/(v_1 \&_1 \mathcal{B}_1[A_1])]\theta_1$. Moreover, if $A_1\theta_1$ is the atom selected in $\mathcal{Q}_1$ by the considered computation rule, we have that $\theta_2 = mgu(\{A_1\theta_1 = H_2\})$. Now, consider $\sigma = mgu(\{A_1 = H_2\})$. Then, the following equalities hold:

$$
\begin{array}{ll}
\theta_1\theta_2 & = \\
\theta_1 mgu(\{A_1\theta_1 = H_2\}) & = (\text{since } Dom(\theta_1) \cap \mathcal{V}ar(\mathcal{R}_2) = \emptyset) \\
\theta_1 mgu(\widehat{mgu}(\{A_1 = H_2\})\theta_1) & = \\
\theta_1 mgu(\hat{\sigma}\theta_1) & = (\text{by Proposition 7}) \\
\theta_1 \Uparrow \sigma & = (\text{by Proposition 7}) \\
\sigma mgu(\widehat{\theta_1}\sigma) & = \\
\sigma mgu(\widehat{mgu}(\{A = H_1\})\sigma) & = (\text{since } Dom(\sigma) \cap \mathcal{V}ar(\mathcal{Q}_0) = \emptyset) \\
\sigma mgu(\{A = H_1\sigma\}) &
\end{array}
$$

Moreover, since $\theta_1\theta_2 \neq fail$, then $\sigma \neq fail$ and thus there exists a rule $\mathcal{R}_3$ obtained by unfolding (atom A_1 in the body of) $\mathcal{R}_1$ by using $\mathcal{R}_2$, such that the head of $\mathcal{R}_3$ is the atom $H_1\sigma$. Now, since $mgu(\{A = H_1\sigma\}) \neq fail$, the following admissible step done on the selected atom A in $\mathcal{Q}_0$ can be proved: $\langle \mathcal{Q}_0; \theta_0 \rangle \rightarrow_{AS1}{}^{\mathcal{R}_3} \langle \mathcal{Q}_3; \theta_0\theta_3 \rangle$, where $\theta_3 = mgu(\{A = H_1\sigma\})$. Finally, since $\theta_1\theta_2 = \sigma\theta_3$, then $\theta_0\theta_1\theta_2 = \theta_0\sigma\theta_3$, and since $Dom(\sigma) \cap \mathcal{V}ar(\mathcal{Q}_0) = \emptyset$

and $Dom(\sigma) \cap Dom(\theta_0) = \emptyset$, we have that $\theta_0\theta_1\theta_2 = \theta_0\theta_3$ $[\mathcal{V}ar(\mathcal{Q}_0)]$, as we wanted to prove.

($\Leftarrow$) This case can be easily proved in a similar way as the previous one, by also exploiting the equivalence between $\theta_1\theta_2$ and $\sigma\theta_3$.

Now, we are able to prove the strong soundness of the transformation.

Theorem 14 (Strong Soundness) *Let $\mathcal{P}$ be a $\mathcal{L}^e$–program and let $\mathcal{Q}$ be a $\mathcal{L}^e$–goal. If $\mathcal{P}'$ is a $\mathcal{L}^e$–program obtained by fuzzy unfolding of $\mathcal{P}$, then, $\langle \mathcal{Q}; id\rangle \rightarrow_{AS}{}^* \langle @(r_1, \ldots, r_n); \theta\rangle$ in $\mathcal{P}$, if $\langle \mathcal{Q}; id\rangle \rightarrow_{AS}{}^* \langle @(r_1, \ldots, r_n); \theta'\rangle$ in $\mathcal{P}'$, where $r_i \in L$ for any $i \in \{1, \ldots, n\}$ and $\theta = \theta'[\mathcal{V}ar(\mathcal{Q})]$.*

PROOF. Let $\mathcal{D}' : [\langle \mathcal{Q}; id\rangle \rightarrow_{AS}{}^* \langle @(r_1, \ldots, r_n); \theta\rangle]$ be the (generic) admissible derivation for $\mathcal{Q}$ in $\mathcal{P}'$ that we plan to simulate by constructing a new derivation $\mathcal{D}$ for $\mathcal{Q}$ in $\mathcal{P}$. The construction of $\mathcal{D}$ is done by induction on the length of $\mathcal{D}'$, k. Since the base case, i.e. $k = 0$, is trivial, we proceed with the general case when $k > 0$. Then, $\mathcal{D}' : [\langle \mathcal{Q}; id\rangle \rightarrow_{AS} \langle \mathcal{Q}'; \vartheta\rangle \rightarrow_{AS}{}^* \langle @(r_1, \ldots, r_n); \theta'\rangle]$. If the first step of $\mathcal{D}'$ has been given with the second or the third rule of Definition 3, or, even it is has been performed with the first one but using a rule also belonging to $\mathcal{P}$, then the claim follows by the inductive hypothesis. Otherwise, this initial step is done with $\rightarrow_{AS1}$ using a rule $\mathcal{R}'$ that has been obtained by unfolding other rule $\mathcal{R} \in \mathcal{P}$. Since the unfolding step has been performed with one of the three rules of Definition 3, we treat each case separately.

(1) Unfolding based on Rule 1.
Let $\mathcal{R} : \langle H_1 \leftarrow_1 \mathcal{B}_1[A_1]; v_1\rangle \in \mathcal{P}$ and $\mathcal{R}_2 : \langle H_2 \leftarrow_2 \mathcal{B}_2; v_2\rangle \in \mathcal{P}$ such that, by unfolding $\mathcal{R}$ w.r.t. $\mathcal{R}_2$ using the first rule in Definition 3, we obtain: $\mathcal{R}' : \langle (H_1 \leftarrow_1 \mathcal{B}_1[A_1/(v_2 \&_2 \mathcal{B}_2)])\sigma; v_1\rangle \in \mathcal{P}'$. If we assume that A is the selected atom in $\mathcal{Q}$ when building derivation $\mathcal{D}'$, then $\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}'} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/(v_2 \&_2 \mathcal{B}_2)])])\sigma\gamma; \sigma\gamma\rangle \rightarrow_{AS}{}^* \langle @(r_1, \ldots, r_n); \theta'\rangle$. Now, the first step of $\mathcal{D}'$ can be simulated in derivation $\mathcal{D}$ by using rules $\mathcal{R}$ and $\mathcal{R}_2$ of $\mathcal{P}$ as follows: $\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1])])\alpha; \alpha\rangle \rightarrow_{AS1}{}^{\mathcal{R}_2} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/(v_2 \&_2 \mathcal{B}_2)])])\alpha\beta; \alpha\beta\rangle$. By Lemma 13 we can conclude that $\alpha\beta = \sigma\gamma[\mathcal{V}ar(\mathcal{Q})]$, and hence the third state in $\mathcal{D}$ coincides syntactically with the second one in $\mathcal{D}'$. Moreover, by the inductive hypothesis $\theta = \theta'[\mathcal{V}ar(\mathcal{Q})]$ and hence the entire admissible derivations $\mathcal{D}$ and $\mathcal{D}'$ are equivalent, as we wanted to prove.

(2) Unfolding based on Rule 2.
Let $\mathcal{R} : \langle H_1 \leftarrow_1 \mathcal{B}_1[A_1]; v_1\rangle \in \mathcal{P}$ and $\mathcal{R}_2 : \langle H_2 \leftarrow; v_2\rangle \in \mathcal{P}$ such that, by unfolding $\mathcal{R}$ w.r.t. $\mathcal{R}_2$ using the second rule in Definition 3, we obtain: $\mathcal{R}' : \langle (H_1 \leftarrow_1 \mathcal{B}_1[A_1/v_2])\sigma; v_1\rangle \in \mathcal{P}'$. Then, $\mathcal{D}'$ has the following form:

$\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}'} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/v_2])])\sigma\gamma; \sigma\gamma\rangle \rightarrow_{AS}{}^{*} \langle @(r_1, \ldots, r_n); \theta'\rangle$.

Now, the first step of $\mathcal{D}'$ can be simulated in derivation $\mathcal{D}$ by using rules $\mathcal{R}$ and $\mathcal{R}_2$ as follows: $\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1])])\alpha; \alpha\rangle \rightarrow_{AS2}{}^{\mathcal{R}_2} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/v_2])])\alpha\beta; \alpha\beta\rangle$. By Lemma 13 we can conclude that $\alpha\beta = \sigma\gamma[\mathcal{V}ar(\mathcal{Q})]$, and hence the third state in $\mathcal{D}$ coincides syntactically with the second one in $\mathcal{D}'$. Moreover, by the inductive hypothesis $\theta = \theta'[\mathcal{V}ar(\mathcal{Q})]$ and hence the entire admissible derivations $\mathcal{D}$ and $\mathcal{D}'$ are equivalent, as we wanted to prove.

(3) Unfolding based on Rule 3.
Let $\mathcal{R} : \langle H_1 \leftarrow_1 \mathcal{B}_1[A_1]; v_1\rangle \in \mathcal{P}$ such that, the selected atom A_1 in $\mathcal{B}_1$ does not unify with the head of any rule in $\mathcal{P}$, and hence, by unfolding $\mathcal{R}$ using the third rule in Definition 3, we obtain the new unfolded rule: $\mathcal{R}' : \langle H_1 \leftarrow_1 \mathcal{B}_1[A_1/\bot]; v_1\rangle \in \mathcal{P}'$. Then, $\mathcal{D}'$ has the following form:
$\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}'} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/\bot])])\alpha; \alpha\rangle \rightarrow_{AS}{}^{*} \langle @(r_1, \ldots, r_n); \theta'\rangle$.
And now, the first step of $\mathcal{D}'$ can be simulated in $\mathcal{P}$ by giving two resolution steps in $\mathcal{D}$: the first one with rule 1 using $\mathcal{R}$ and the second one with rule 3. That is: $\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1])])\alpha; \alpha\rangle \rightarrow_{AS3} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/\bot])])\alpha; \alpha\rangle$. Since this last state coincides syntactically with the second one in $\mathcal{D}'$, our claim holds by the inductive hypothesis.

Now, we proceed with the counterpart of the previous Theorem, that is, the strong completeness of the fuzzy unfolding transformation.

Theorem 15 (Strong Completeness) *Let $\mathcal{P}$ be a $\mathcal{L}^e$–program and let $\mathcal{Q}$ be a $\mathcal{L}^e$–goal. If $\mathcal{P}'$ is a $\mathcal{L}^e$–program obtained by fuzzy unfolding of $\mathcal{P}$, then, $\langle \mathcal{Q}; id\rangle \rightarrow_{AS}{}^{*} \langle @(r_1, \ldots, r_n); \theta'\rangle$ in $\mathcal{P}'$, if $\langle \mathcal{Q}; id\rangle \rightarrow_{AS}{}^{*} \langle @(r_1, \ldots, r_n); \theta\rangle$ in $\mathcal{P}$, where $r_i \in L$ for any $i \in \{1, \ldots, n\}$ and $\theta' = \theta[\mathcal{V}ar(\mathcal{Q})]$.*

PROOF. Our proof consists in simulating in $\mathcal{P}'$ a re–ordered admissible derivation originally performed in $\mathcal{P}$. So, consider the following (generic) k–steps admissible derivation for $\mathcal{Q}$ in $\mathcal{P}$, $\mathcal{D}_0 : [\langle \mathcal{Q}; id\rangle \rightarrow_{AS}{}^{k} \langle @(r_1, \ldots, r_n); \theta_0\rangle]$. Assume now that $\mathcal{R} \in \mathcal{P}$ is the rule unfolded in $\mathcal{P}$ which obviously does not belong to $\mathcal{P}'$. Any existing step done with rule $\mathcal{R}$ in $\mathcal{D}_0$ introduces an instance of the body of $\mathcal{R}$ in the next state of the derivation. Since we are dealing which an admissible successful derivation, this *instanced* body of $\mathcal{R}$ must necessarily be reduced in the immediately next step or in subsequent ones. For the second case, we can safely interchange the step done with rule $\mathcal{R}$ and the next one, by application of the Switching Lemma 9. Moreover, by repeated application of this Lemma, we can obtain a new k–steps admissible derivation $\mathcal{D} : [\langle \mathcal{Q}; id\rangle \rightarrow_{AS}{}^{k} \langle @(r_1, \ldots, r_n); \theta\rangle]$ in $\mathcal{P}$ verifying $\theta = \theta_0[\mathcal{V}ar(\mathcal{Q})]$, where any step (if it exists) using the rule $\mathcal{R}$ unfolded in $\mathcal{P}$, is followed by other step exploiting an atom just introduced by the previous step (i.e., belonging to the instanced body of $\mathcal{R}$). We say that $\mathcal{D}$ is an admissible derivation re–ordered

w.r.t. rule $\mathcal{R}$.

Now, and similarly to the previous theorem, we are going to simulate $\mathcal{D}$ in $\mathcal{P}'$ by constructing a new derivation $\mathcal{D}'$ using the rules of $\mathcal{P}'$ and following an schema perfectly analogous to the one used in Theorem 14, but inverting now the use of terms $\mathcal{P}$ and $\mathcal{P}'$ (and related ones). The construction of $\mathcal{D}'$ is done by induction on the length of $\mathcal{D}$, k. Since the case base, i.e. $k = 0$, is trivial, we proceed with the general case when $k > 0$. Then, $\mathcal{D} : [\langle \mathcal{Q}; id\rangle \rightarrow_{AS} \langle \mathcal{Q}'; \vartheta\rangle \rightarrow_{AS}{}^{*} \langle @(r_1, \ldots, r_n); \theta\rangle]$. If the first step of $\mathcal{D}$ has been given with the second or third rule of Definition 3, or, even it is has been performed with the first one but using a rule also belonging to $\mathcal{P}'$, then the claim follows by the inductive hypothesis. Otherwise, this initial step is done with $\rightarrow_{AS1}$ using a rule $\mathcal{R}$ that, once it is unfolded, generates the new transformed rule $\mathcal{R}' \in \mathcal{P}'$. Since the unfolding step has been performed with one of the three rules of Definition 3, we treat each case separately.

(1) Unfolding based on Rule 1.
Let $\mathcal{R} : \langle H_1 \leftarrow_1 \mathcal{B}_1[A_1]; v_1\rangle \in \mathcal{P}$ and $\mathcal{R}_2 : \langle H_2 \leftarrow_2 \mathcal{B}_2; v_2\rangle \in \mathcal{P}$ such that, by unfolding $\mathcal{R}$ w.r.t. $\mathcal{R}_2$ using the first rule in Definition 3, we obtain $\mathcal{R}' : \langle (H_1 \leftarrow_1 \mathcal{B}_1[A_1/(v_2 \&_2 \mathcal{B}_2)])\sigma; v_1\rangle \in \mathcal{P}'$. Assuming that A is the selected atom in $\mathcal{Q}$ when building derivation $\mathcal{D}$, and having into account that $\mathcal{D}$ is an admissible derivation reordered w.r.t. rule $\mathcal{R}$, then it has the the following form: $\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1])])\alpha; \alpha\rangle \rightarrow_{AS1}{}^{\mathcal{R}_2} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/(v_2 \&_2 \mathcal{B}_2)])])\alpha\beta; \alpha\beta\rangle \rightarrow_{AS}{}^{*} \langle @(r_1, \ldots, r_n); \theta\rangle$. Now, the first two steps of $\mathcal{D}$ can be simulated in $\mathcal{P}'$ by using rule $\mathcal{R}'$ in derivation $\mathcal{D}'$ as follows: $\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}'} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/(v_2 \&_2 \mathcal{B}_2)])])\sigma\gamma; \sigma\gamma\rangle \rightarrow_{AS}{}^{*} \langle @(r_1, \ldots, r_n); \theta'\rangle$. By Lemma 13 we have that $\alpha\beta = \sigma\gamma[\mathcal{V}ar(\mathcal{Q})]$, and hence the third state in $\mathcal{D}$ coincides syntactically with the second one in $\mathcal{D}'$. Moreover, by the inductive hypothesis $\theta = \theta'[\mathcal{V}ar(\mathcal{Q})]$ and hence the entire admissible derivations $\mathcal{D}$ and $\mathcal{D}'$ are equivalent, as we wanted to prove.

(2) Unfolding based on Rule 2.
Let $\mathcal{R} : \langle H_1 \leftarrow_1 \mathcal{B}_1[A_1]; v_1\rangle \in \mathcal{P}$ and $\mathcal{R}_2 : \langle H_2 \leftarrow; v_2\rangle \in \mathcal{P}$ such that, by unfolding $\mathcal{R}$ w.r.t. $\mathcal{R}_2$ using the second rule in Definition 3, we obtain: $\mathcal{R}' : \langle (H_1 \leftarrow_1 \mathcal{B}_1[A_1/v_2])\sigma; v_1\rangle \in \mathcal{P}'$. Since $\mathcal{D}$ is an admissible derivation reordered w.r.t. rule $\mathcal{R}$, and assuming that A is the selected atom in $\mathcal{Q}$, then $\mathcal{D}$ has the form: $\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1])])\alpha; \alpha\rangle \rightarrow_{AS2}{}^{\mathcal{R}_2} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/v_2])])\alpha\beta; \alpha\beta\rangle \rightarrow_{AS}{}^{*} \langle @(r_1, \ldots, r_n); \theta\rangle$. And now, the first two steps in $\mathcal{D}$ can be simulated in $\mathcal{P}'$ by using rule $\mathcal{R}'$ as follows: $\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}'} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/v_2])])\sigma\gamma; \sigma\gamma\rangle \rightarrow_{AS}{}^{*} \langle @(r_1, \ldots, r_n); \theta'\rangle$. By Lemma 13 we can conclude that $\alpha\beta = \sigma\gamma[\mathcal{V}ar(\mathcal{Q})]$, and hence the third state in $\mathcal{D}$ coincides syntactically with the second one in $\mathcal{D}'$. Moreover, by the inductive hypothesis $\theta = \theta'[\mathcal{V}ar(\mathcal{Q})]$ and hence the entire admissible derivations $\mathcal{D}$ and $\mathcal{D}'$ are equivalent, as we wanted to prove.

(3) Unfolding based on Rule 3.
Let $\mathcal{R} : \langle H_1 \leftarrow_1 \mathcal{B}_1[A_1]; v_1\rangle \in \mathcal{P}$ such that, the selected atom A_1 in $\mathcal{B}_1$ does not unify with the head of any rule in $\mathcal{P}$, and hence, by unfolding $\mathcal{R}$ using the third rule in Definition 3, we obtain the new unfolded rule $\mathcal{R}' : \langle H_1 \leftarrow_1 \mathcal{B}_1[A_1/\bot]; v_1\rangle \in \mathcal{P}'$. Assuming that A is the selected atom in $\mathcal{Q}$ when building derivation $\mathcal{D}$, and having into account that $\mathcal{D}$ is an admissible derivation reordered w.r.t. rule $\mathcal{R}$, then it has the following form: $\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1])])\alpha; \alpha\rangle \rightarrow_{AS3} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/\bot])])\alpha; \alpha\rangle \rightarrow_{AS}{}^{*} \langle @(r_1, \ldots, r_n); \theta\rangle$. And now, the first two steps of $\mathcal{D}$ can be simulated in derivation $\mathcal{D}'$ by using rule $\mathcal{R}'$ as follows: $\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}'} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/\bot])])\alpha; \alpha\rangle$. Since this last state coincides syntactically with the third one in $\mathcal{D}$, our claim holds by the inductive hypothesis.

Now, we are able to formalize and prove the best properties of fuzzy unfolding (namely, its strong correctness and the guarantee that it produces improvements on transformed programs) as follows:

Theorem 16 (Strong Correctness of Fuzzy Unfolding) *Let $\mathcal{P}$ be a $\mathcal{L}^e$–program, and let $\mathcal{Q}$ be a $\mathcal{L}^e$–goal. If $\mathcal{P}'$ is a $\mathcal{L}^e$–program obtained by fuzzy unfolding of $\mathcal{P}$, then,*
$\langle \mathcal{Q}; id\rangle \rightarrow_{AS}{}^{n} \langle @(r_1, \ldots, r_k); \theta\rangle$ *in* $\mathcal{P}$ *iff* $\langle \mathcal{Q}; id\rangle \rightarrow_{AS}{}^{m} \langle @(r_1, \ldots, r_k); \theta'\rangle$ *in* $\mathcal{P}'$, *where* $r_i \in L$ *for any* $i \in \{1, \ldots, k\}$, $\theta = \theta'[\mathcal{V}ar(\mathcal{Q})]$ *and* $m \leq n$.

PROOF. The two claims of Theorem 16 can be easily proved as follows:

- The strong correctness of the transformation, i.e., the equivalence of fuzzy computed answers obtained when executing a goal w.r.t. original and unfolded programs, is immediate by simply applying Theorems 14 and 15.
- Regarding the reduction of the length of admissible derivations in transformed programs, we have seen in proofs of both theorems 14 and 15 that any fuzzy admissible step done with a new rule, obtained after applying fuzzy unfolding, subsumes two fuzzy admissible steps done with rules of the original program, which confirms that $m \leq n$, as we wanted to prove.

7 Conclusions

This work introduces a safe transformation rule for unfolding fuzzy multi–adjoint logic programs. To the best of our knowledge, this is the first time this issue, of integrating transformation techniques in the context of fuzzy multi–adjoint logic languages, is treated in the literature. We have defined the notion

of fuzzy unfolding of multi–adjoint logic programs (Definition 11) and we have demonstrated the (strong) correctness (Theorem 16) of the transformation rule as well as its capability to produce significant improvements on the final code.

It is well–known in the literature on program transformation that since unfolding generally increases the size of residual programs, more (secondary and heap) memory is needed for storing them. However, the stack memory required at execution time usually decreases, mainly due to the reduction in the length of derivations (in declarative programing, there is a direct correspondence between derivation states and elements pushed on the system stack). These facts justify the benefits (not only in time, but also in space) of using unfolding.

As an additional advantage, transformation sequences built by only using unfolding, can be guided in a blind way since any transformation step always produces an improvement on transformed programs. This contrasts with other transformation rules, such as definition introduction or folding, whose usage may degrade the efficiency of programs, if appropriate "transformation strategies" are not used to drive the construction of the transformation sequence. On the other hand, it is well established that certain transformation strategies, such as composition or tupling, are able to change the original program algorithmic complexity and to obtain even super–linear speedups. Unfolding can be combined jointly with other transformation rules and using transformation strategies to increase its power.

The results in this paper can be thought as a basis to optimize fuzzy multi–adjoint logic programs and they are the first step in the construction of a fold/unfold framework for optimizing this class of programs.

Acknowledgment

We are very grateful to anonymous reviewers for providing us worthy material and suggestive discussions on fuzzy logic programming which helped us to highly improve this paper.

We are also grateful to P. Vojtáš and M. Ojeda-Aciego to allow us free access to their works.

References

[1] H. Nguyen, E. Walker, A First Course in Fuzzy Logic, Chapman & Hall/CRC, Boca Ratón, Florida, 2000.

[2] J. Lloyd, Foundations of Logic Programming, Springer-Verlag, Berlin, 1987, second edition.

[3] F. Arcelli, F. Formato, Likelog: A logic programming language for flexible data retrieval, in: Proc. of the 1999 ACM Symposium on Applied Computing (SAC'99), San Antonio, Texas, USA, ACM, New York, 1999, pp. 260–267.

[4] P. Vojtáš, L. Paulík, Soundness and completeness of non-classical extended SLD-resolution, in: R. D. et al (Ed.), Proc. of Extensions of Logic Programming, ELP'96, Leipzig, Springer Verlag, LNCS 1050, 1996, pp. 289–301.

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

[6] J. Medina, M. Ojeda-Aciego, P. V. s, Multi-adjoint logic programming with continuous semantics, in: Proc. of Logic Programming and Non-Monotonic Reasoning, LPNMR'01, Springer-Verlag, LNAI 2173, 2001, pp. 351–364.

[7] J. Medina, M. Ojeda-Aciego, P. Vojtáš, A procedural semantics for multi-adjoint logic programming, in: Proc. of Progress in Artificial Intelligence, EPIA'01, Springer-Verlag, LNAI 2258, 2001, pp. 290–297.

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

[9] S. Guadarrama, S. Muñoz, C. Vaucheret, Fuzzy prolog: A new approach using soft constraints propagation, Fuzzy Sets and Systems 144 (1) (2004) 127–150.

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

[11] R. Burstall, J. Darlington, A Transformation System for Developing Recursive Programs, Journal of the ACM 24 (1) (1977) 44–67.

[12] H. Tamaki, T. Sato, Unfold/Fold Transformations of Logic Programs, in: S. Tärnlund (Ed.), Proc. of Second Int'l Conf. on Logic Programming, 1984, pp. 127–139.

[13] M. Alpuente, M. Falaschi, G. Moreno, G. Vidal, Rules + Strategies for Transforming Lazy Functional Logic Programs, Theoretical Computer Science 311 (2004) 479–525.

[14] A. Pettorossi, M. Proietti, A Comparative Revisitation of Some Program Transformation Techniques, in: O. Danvy, R. Glück, P. Thiemann (Eds.), Partial Evaluation, Int'l Seminar, Dagstuhl Castle, Germany, Springer LNCS 1110, 1996, pp. 355–385.

[15] P. Julian, G. Moreno, J. Penabad, Unfolding Fuzzy Logic Programs, in: Proc. of the Fourth International Conference on Intelligent Systems Design and Applications, ISDA'04, Budapest, Hungary, (Sponsored by IEEE), 2004, pp. 595–600.

[16] J.L. Lassez, M. J. Maher, K. Marriott, Unification Revisited, in: J. Minker (Ed.), Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, Los Altos, Ca., 1988, pp. 587–625.

[17] C. Palamidessi, Algebraic Properties of Idempotent Substitutions, in: M. Paterson (Ed.), Proc. of the 17th Int'l Colloquium on Automata, Languages and Programming, Springer LNCS 443, 1990, pp. 386–399.

[18] P. Julian, C. Villamizar, Analizing Definitional Trees: Looking for Determinism, in: Y. Kameyama, M. P. J. Stuckey (Eds.), Proc. of the 7th Fuji International Symposium on Functional and Logic Programming, FLOPS'04, Nara, Japan, Springer-Verlag, LNCS 2998, 2004, pp. 55–69.