Similarity-Based Unification: A Multi-Adjoint Approach

Jesús Medina, Manuel Ojeda-Aciego[1]

Dept. Matemática Aplicada. Universidad de Málaga
{jmedina,aciego}@ctima.uma.es

Peter Vojtáš[2]

Institute of Informatics. P.J. Šafárik University
vojtas@kosice.upjs.sk

Abstract

The aim of this paper is to build a formal model for similarity-based fuzzy unification in multi-adjoint logic programs. Specifically, a general framework of logic programming which allows the simultaneous use of different implications in the rules and rather general connectives in the bodies is introduced, then a procedural semantics for this framework is presented, and an aproximative-completeness theorem proved. On this computational model, a similarity-based unification approach is constructed by simply adding axioms of fuzzy similarities and using classical crisp unification which provides a semantic framework for logic programming with different notions of similarity.

Key words: Similarity, fuzzy unification.

1 Introduction

It is usual practice in mathematical logic for a formal model to have clearly defined its syntactical part (which deals with proofs) and its semantical part (dealing with truth and/or satisfaction). When applying logic to computer science, mainly in logic programming, a different terminology is used, and we speak about the declarative part of the formal model (corresponding to truth

[1] Partially supported by Spanish DGI project BFM2000-1054-C02-02.
[2] Partially supported by Slovak project VEGA 1/7557/20

and satisfaction) and the procedural part, more focused on algorithmic aspects of finding proofs (automated deduction).

Unification is an important part of procedural semantics for many formal models, because it helps to identify instantiations of different statements, that is, to make them syntactically (letter by letter) equal; and unified (identical) statements can be handled identically. In the classical case, on the one hand, there was no need to clearly specify the declarative and procedural parts of a formal model of unification because, on the declarative part, there was the requirement of being syntactically equal (and hence equal also from the point of view of truth and satisfaction); on the other hand, the procedural part of classical unification is well developed, namely, different unification algorithms have been studied. This is no longer case when considering fuzzy unification, where both the declarative and the procedural part of a formal model of fuzzy unification are needed.

We mention below some possible examples of problems from the real world, whose solution needs an adequate treatment of similarity, but one can imagine many other syntactical, linguistic and conceptual sources, see e.g. [9]. For illustration purposes consider the following situations:

- Firstly, imagine a hotel name "Salaš" in Slovak, which can in Hungarian pronunciation can be written as "Szálás", in Polish "Szałas" and in English (omitting the "check sign" from Slovak) as "Salas". Does the query using "Salaš" unify with a database fact about "Szałas"?, do subgoals about "Szálás" and "Salas" fit together? A common-sense intuition suggests that, without the knowledge of any additional fact, the system itself cannot glue together information about objects with different syntactical form. So the main question is how to describe this additional information.
- Another source of problems comes from the need of inter-operability, in which a client requires apparently homogeneous access to heterogeneous servers. This heterogeneity causes, for instance, that web users accessing these information sources usually require a multi-step process utilizing the intelligence of the end-user to navigate and to resolve heterogeneity by applying several similarity criteria [7].
- A third interesting example comes again from internet, whose initial design was made as an initiative for connecting sites containing information stored for direct human processing, but its next generation (the semantic web) is aimed at storing machine-processable information. For instance, implementing search engines which use ontologies to find pages with words that are syntactically different but semantically similar [3].
- Our final motivating example has to do with multimedia database queries. A user might want to query a multimedia database system over several properties he/she is interested in. For instance, consider the case that one is searching for a movie clip that has a predominantly red scene with a loud

noise in the sound track, this example comes from [5]. There is likely to be a score giving the redness of the scene and a different score giving the loudness of the sound, and these two scores have to be somehow aggregated to be able to obtain their similarity degree and provide an ordered list of results.

Several approaches have been proposed for dealing with these problems; in this paper we choose the way of including additional information about fuzzy similarities of different objects and using axioms of equality to transfer properties between these objects, our main aim being not to give practical advice on how to detect and handle similarities in practical applications, but to give a formal model for both the declarative and the procedural part of similarity-based fuzzy unification. Here we stress the fact that, for fuzzy unification, both a procedural and declarative semantics are needed, as opposed to the two valued case.

Our approach to fuzzy unification is based on a theory of fuzzy logic programming with crisp unification constructed on the multi-adjoint framework recently introduced in [12,13]. We recall definitions of declarative and procedural semantics of multi-adjoint logic programming and show how its soundness and completeness, and especially the fix-point theorem and the minimal model obtained by the iteration of the immediate consequences operator, can give a base for a sound and complete model of fuzzy unification. The fact that this theory of fuzzy unification is developed inside the realm of fuzzy logic programming is very important for later integration of fuzzy similarity-based unification and fuzzy logic programming deduction.

The structure of the paper is as follows: in Sections 2 and 3, a general theory of logic programming which allows the simultaneous use of different implications in the rules and rather general connectives in the bodies can be constructed. Models of multi-adjoint logic programs are postfix-points of the immediate consequences operator, which is proved to be monotonic under very general hypotheses. The continuity of the immediate consequences operator is proved under the general assumption of continuity of all the operators in the program (but, possibly, implications). Later, in Section 4, a procedural semantics for the general theory of multi-adjoint logic programming is presented, in the spirit of [13], and a quasi-completeness theorem is proved. In Section 5, our similarity-based approach to unification is introduced. Roughly speaking, we add axioms of fuzzy similarities and, using the computational model provided by the procedural semantics which uses classical crisp unification, we provide a semantic framework for logic programming with different notions of similarity. Finally, we give some comparisons with other approaches. It is worth to mention that the weak unification algorithm introduced in [15] can be completely emulated by our similarity-based unification model.

2 Multi-adjoint logic programming

The intuition behind multi-adjoint logic programs is as follows: Considering different implication operators, such as Łukasiewicz, Gödel or product implication in the same logic program, naturally leads to the allowance of several adjoint pairs in the lattice of truth-values. This idea is used in [12] to introduce multi-adjoint logic programs as an extension of monotonic and residuated logic programs, presented in [2], so that it is possible to use a number of different implications in the rules of our programs. Specifically, the language and semantics of definite logic programs are generalized in order to encompass more complex rules.

The semantical framework of multi-adjoint logic programs is based on the so-called multi-adjoint lattices. The semantic basis of the notion of consequence in generalized logic programs is that of *adjoint pair*, which allows that fairly general conjunctors and their adjoints are used as, for instance, in [17].

The concept of adjoint pair was firstly introduced in a logical context by Pavelka [14], who interpreted the poset structure of the set of truth-values as a category, and the relation between the connectives of implication and conjunction as functors in this category. The result turned out to be another example of the well-known concept of adjunction, introduced by Kan in the general setting of category theory in 1950.

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

(1) Operation $\&$ is increasing in both arguments, i.e. if $x_1, x_2, y \in P$ such that $x_1 \preceq x_2$ then $(x_1 \mathbin{\&} y) \preceq (x_2 \mathbin{\&} y)$ and $(y \mathbin{\&} x_1) \preceq (y \mathbin{\&} x_2)$;
(2) Operation $\leftarrow$ is increasing in the first argument (the consequent) and decreasing in the second argument (the antecedent), i.e. if $x_1, x_2, y \in P$ such that $x_1 \preceq x_2$ then $(x_1 \leftarrow y) \preceq (x_2 \leftarrow y)$ and $(y \leftarrow x_2) \preceq (y \leftarrow x_1)$;
*(3) **Adjoint property.** For any $x, y, z \in P$, we have that $x \preceq (y \leftarrow z)$ holds if and only if $(x \mathbin{\&} z) \preceq y$ holds.*

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

The main point in the extension of the results in [2,17,18] to a more general setting, in which different implications (Łukasiewicz, Gödel, product) and several modus ponens-like inference rules are used, naturally leads to considering several adjoint pairs in the residuated lattice, leading to what we call a *multi-adjoint* lattice. More formally,

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

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

Note that residuated lattices [4] are a special case of multi-adjoint lattice, in which only one adjoint pair is present, and the underlying lattice has monoidal structure wrt $\&$ and $\top$ (in general we are assuming neither associativity nor commutativity of $\&$).

From the point of view of expressiveness, it is convenient to allow extra operators to be involved with the operators in the multi-adjoint lattice. The structure which captures this possibility is that of a multi-adjoint algebra.

Definition 3 *Let* Ω *be a graded set containing operators* $\leftarrow_i$ *and* $\&_i$ *for* $i = 1, \dots, n$ *and possibly some extra operators, and let* $\mathfrak{L} = \langle L, I \rangle$ *be an* Ω*-algebra whose carrier set* L *is a complete lattice under* $\preceq$*.*

We say that $\mathfrak{L}$ *is a* multi-adjoint Ω-algebra *with respect to the pairs* $(\leftarrow_i, \&_i)$ *for* $i = 1, \dots, n$ *if* $\mathcal{L} = (L, \preceq, I(\leftarrow_1), I(\&_1), \dots, I(\leftarrow_n), I(\&_n))$ *is a multi-adjoint lattice.*

In the following, as we will work with a fixed graded set Ω, the prefix Ω- will be dropped and, when necessary, we will talk simply about algebras.

3 Syntax and Semantics of Multi-Adjoint Programs

The definition of multi-adjoint logic program is given, as usual in fuzzy logic programming, as a set of weighted rules and facts of a given first-order language $\mathfrak{F}$, constructed as an algebra. Note that we will be allowed to use different implications in our rules.

Definition 4 *A* multi-adjoint logic program *is a set* $\mathbb{P}$ *of weighted rules of the form* $\langle A \leftarrow_i \mathcal{B}, \vartheta \rangle$ *such that:*

(1) The consequent of the implication, A*, is an atom which is called the* head*.*
(2) The antecedent of the implication, $\mathcal{B}$*, is called the* body*, and is a formula built from atoms* $B_1, \dots, B_n$ $(n \geq 0)$ *by the use of conjunctors, disjunctors, and aggregators.*
(3) The confidence factor ϑ *is an element (a truth-value) of* L*.*

As usual, facts *are rules with body* $\top$*,* goals *or* queries *are atoms intended as questions* $?A$ *prompting the system. Free occurrences of variables in the program are assumed to be universally quantified.*

Definition 5 *An* interpretation *is a mapping from the Herbrand base, $B_{\mathbb{P}}$, of the program $\mathbb{P}$ to the multi-adjoint lattice of truth-values $\langle L, \preceq \rangle$.*

By using the unique homomorphic extension, it is possible to extend uniquely the mapping I, defined on $B_{\mathbb{P}}$, to be defined on the set of all ground formulas of the language, this extension will be denoted $\hat{I}$. The extension for the non-ground case is also straightforward, due to the fact that all our formulas are considered universally closed; this way, for a non-ground formula A, the interpretation I is defined as follows:[3]

$$\hat{I}(A) = \inf_{\xi}\{\hat{I}(A\xi) \mid A\xi \text{ is a ground instantiation of } A\ \}$$

The ordering $\preceq$ in L can be easily extended to the set of interpretations as usual, $I_1 \sqsubseteq I_2$ iff $I_1(A) \preceq I_2(A)$ for all ground atom. The least interpretation $\vartriangle$ maps every ground atomic formula to the least element $\bot$ of L.

Definition 6 *An interpretation I* satisfies *a weighted rule $\langle A \leftarrow_i \mathcal{B}, \vartheta \rangle$, if and only if $\vartheta \preceq \hat{I}(A \leftarrow_i \mathcal{B})$. An interpretation I is a* model *of a multi-adjoint logic program $\mathbb{P}$ iff all weighted rules in $\mathbb{P}$ are satisfied by I.*

Note that we will be working with two algebras, $\mathfrak{F}$ for the formulas and $\mathfrak{L}$ for its interpretations, to avoid the risk of confusion we will introduce a special notation to clarify which algebra an operator belongs to. Let ω be an operator symbol, its interpretation under $\mathfrak{L}$ is denoted $\dot{\omega}$ (a dot on the operator), whereas ω itself will be interpreted in $\mathfrak{F}$.

Note the following equalities, where ξ is ground

$$\hat{I}((A \leftarrow_i \mathcal{B})\xi) = \hat{I}(A\xi \leftarrow_i \mathcal{B}\xi) = \hat{I}(A\xi) \dot{\leftarrow}_i \hat{I}(\mathcal{B}\xi) = I(A\xi) \dot{\leftarrow}_i \hat{I}(\mathcal{B}\xi)$$

where the evaluation of $\hat{I}(\mathcal{B}\xi)$ proceeds inductively as usual. Similarly, a fact $\langle A \leftarrow_i \top, \vartheta \rangle$ is satisfied if

$$\vartheta \preceq \hat{I}((A \leftarrow_i \top)) \preceq I((A \leftarrow_i \top)\xi) = I(A) \dot{\leftarrow}_i \top$$

now, by the adjoint property, this is equivalent to $\vartheta \mathbin{\dot{\&}_i} \top \preceq I(A)$ and, by the third assumption in Definition 2, this gives $\vartheta \preceq I(A)$.

Definition 7 *A pair $(\lambda; \theta)$ where $\lambda \in L$ and θ is a substitution, is a* correct answer *for a program $\mathbb{P}$ and a query $?A$ if for any model of $\mathbb{P}$ we have*

$$\lambda \preceq \inf_{\xi} \left\{ I(A\theta\xi) \right\} = \hat{I}(A\theta)$$

[3] In the rest of the paper, we will always use the notation $\inf_{\xi}$ to denote the infimum on the set of all ground instantiations of the corresponding formula.

The immediate consequences operator, given by van Emden and Kowalski, can be generalized to the framework of multi-adjoint logic programs as follows:

Definition 8 *Let $\mathbb{P}$ be a multi-adjoint logic program. The* immediate consequences operator $T_{\mathbb{P}}$ *maps interpretations to interpretations and is defined in such a way that, given an interpretation I and a ground atom A,*

$$T_{\mathbb{P}}(I)(A) = \sup\left\{\vartheta \mathbin{\dot{\&}_i} \hat{I}(\mathcal{B}\theta) \mid \langle C \leftarrow_i \mathcal{B}, \vartheta\rangle \in \mathbb{P} \text{ and } A = C\theta\right\}$$

As it is usual in the logic programming framework, the semantics of a multi-adjoint logic program is defined as the least fix-point of $T_{\mathbb{P}}$. In [12], the monotonicity of $T_{\mathbb{P}}$, and its continuity (granted under continuity of all the operators in the body) were proved in the propositional case. These results about monotonicity and continuity of $T_{\mathbb{P}}$ are extended to first-order multi-adjoint logic programs in the rest of the section.

Theorem 9 *An interpretation I is a model of a multi-adjoint logic program $\mathbb{P}$ iff $T_{\mathbb{P}}(I) \sqsubseteq I$.*

PROOF. Firstly, assume that I is a model for $\mathbb{P}$, and let us prove that $T_{\mathbb{P}}(I) \sqsubseteq I$.

Let A be a ground atom, for all weighted rule $\langle C \leftarrow_i \mathcal{B}, \vartheta\rangle$ matching with A (that is, there is a ground instantiation $(C \leftarrow_i \mathcal{B})\theta$ such that $A = C\theta$) we have, by hypothesis, the following chain of (in)equalities:

$$\vartheta \preceq \inf_{\xi}\{\hat{I}((C \leftarrow_i \mathcal{B})\xi)\} \preceq \hat{I}((C \leftarrow_i \mathcal{B})\theta) = \hat{I}(C\theta \leftarrow_i \mathcal{B}\theta) = \hat{I}(C\theta) \mathrel{\dot{\leftarrow}_i} \hat{I}(\mathcal{B}\theta)$$

now, applying the adjoint property, we have $\vartheta \mathbin{\dot{\&}_i} \hat{I}(\mathcal{B}\theta) \preceq I(C\theta) = I(A)$ and taking suprema on the corresponding set of substitutions we get

$$\sup\{\vartheta \mathbin{\dot{\&}_i} \hat{I}(\mathcal{B}\theta) \mid \langle C \leftarrow_i \mathcal{B}, \vartheta\rangle \in \mathbb{P} \text{ and } A = C\theta\} \preceq I(A)$$

If there is no rule with head A, then

$$T_{\mathbb{P}}(I)(A) = \sup \varnothing = \bot \preceq I(A)$$

We have also to prove the converse: if $T_{\mathbb{P}}(I)(A) \preceq I(A)$ for all ground atom A, then I is a model for $\mathbb{P}$. This is immediate, as for any weighted rule $\langle C \leftarrow_i \mathcal{B}, \vartheta\rangle$

in $\mathbb{P}$ and ground instantiation ξ for $C \leftarrow_i \mathcal{B}$; by definition of $T_{\mathbb{P}}$, we have that $\vartheta \,\&_i\, \hat{I}(\mathcal{B}\xi) \preceq T_{\mathbb{P}}(I)(C\xi)$. Now, by hypothesis,

$$\vartheta \,\&_i\, \hat{I}(\mathcal{B}\xi) \preceq I(C\xi) \;\text{ for all ground instantiation } \xi$$

and, again by the adjoint property, $\vartheta \preceq I(C\xi) \leftarrow_i \hat{I}(\mathcal{B}\xi) = \hat{I}((C \leftarrow_i \mathcal{B})\xi)$ and finally, taking the infimum:

$$\vartheta \preceq \inf_{\xi}\{\hat{I}((C \leftarrow_i \mathcal{B})\xi)\}$$

The monotonicity of the operator $T_{\mathbb{P}}$ is given below:

Theorem 10 *The operator $T_{\mathbb{P}}$ is monotonic.*

PROOF. Consider I and J two interpretations such that $I \sqsubseteq J$, and let us prove that $T_{\mathbb{P}}(I) \sqsubseteq T_{\mathbb{P}}(J)$.

Firstly, let us prove that $\hat{I}(F) \preceq \hat{J}(F)$ for all ground body formula F. We will use structural induction:

If F is a ground atomic formula, then it is obvious, ie

$$\hat{I}(F) = I(F) \preceq J(F) = \hat{J}(F)$$

For the inductive case, consider F to be a body formula $@(F_1, \dots, F_n)$ and assume that $\hat{I}(F_i) \preceq \hat{J}(F_i)$ for all $i = 1, \dots, n$. By definition, @ behaves as an aggregator, and therefore, using the induction hypothesis

$$\hat{I}(F) = \dot{@}(\hat{I}(F_1), \dots, \hat{I}(F_n)) \preceq \dot{@}(\hat{J}(F_1), \dots, \hat{J}(F_n)) = \hat{J}(F)$$

With the previous result, let us finish the proof:

Consider a rule $\langle C \leftarrow_i \mathcal{B}, \vartheta\rangle$ in $\mathbb{P}$ and a ground θ with $A = C\theta$; by the induction hypotheses, as $\mathcal{B}\theta$ is a ground formula, we have the inequality $\hat{I}(\mathcal{B}\theta) \preceq \hat{J}(\mathcal{B}\theta)$, then we also have $\vartheta \,\dot{\&}_i\, \hat{I}(\mathcal{B}\theta) \preceq \vartheta \,\dot{\&}_i\, \hat{J}(\mathcal{B}\theta)$ for all i, since operators $\dot{\&}_i$ are increasing. Now, by taking suprema $T_{\mathbb{P}}(I)(A) \preceq T_{\mathbb{P}}(J)(A)$ for all A.

Due to the monotonicity of the immediate consequences operator, the semantics of $\mathbb{P}$ is given by its least model which, by Knaster-Tarski's theorem together with Theorem 9, is exactly the least fix-point of $T_{\mathbb{P}}$, which can be obtained by transfinitely iterating $T_{\mathbb{P}}$ from the least interpretation Δ.

A first result in this approach is that whenever every operator turns out to be continuous in the lattice, then $T_{\mathbb{P}}$ is also continuous and, consequently, its least fix-point can be obtained by a countably infinite iteration from the least interpretation.

Let us state the definition of continuous function which will be used.

Definition 11 *Let L be a complete lattice and let $f\colon L \to L$ be a mapping. We say that f is* continuous *if it preserves suprema of directed sets, that is, given a directed set X one has*

$$f(\sup X) = \sup\{f(x) \mid x \in X\}$$

A mapping $g\colon L^n \to L$ is said to be continuous *provided that it is continuous in each argument separately.*

Let $\mathfrak{F}$ be a language interpreted on a multi-adjoint algebra $\mathfrak{L}$, and let ω be any operator symbol in the language. We say that ω is continuous *if its interpretation under $\mathfrak{L}$, that is $\dot{\omega}$, is continuous in L.*

Now we state and prove a technical lemma which will allow us to prove the continuity of the immediate consequences operator.

Lemma 12 *Let $\mathbb{P}$ be a multi-adjoint program, and let $\mathcal{B}$ be any body formula in $\mathbb{P}$. Assume that all the operators in $\mathcal{B}$ are continuous, let X be a directed set of interpretations, and write $S = \sup X$; then*

$$\hat{S}(\mathcal{B}) = \sup\{\hat{J}(\mathcal{B}) \mid J \in X\}$$

PROOF. Firstly, let us prove by structural induction the equality for all ground formula $\mathcal{B}$; in the base case of being an atomic ground formula the result is obvious, since

$$\hat{S}(\mathcal{B}) = S(\mathcal{B}) = \sup\{J(\mathcal{B}) \mid J \in X\} = \sup\{\hat{J}(\mathcal{B}) \mid J \in X\}$$

For the inductive case, consider $\mathcal{B} = @(\mathcal{B}_1, \dots, \mathcal{B}_n)$, then by definition we have $\hat{S}(\mathcal{B}) = \dot{@}(\hat{S}(\mathcal{B}_1), \dots, \hat{S}(\mathcal{B}_n))$ since $\mathcal{B}$ is a ground formula. Now, by the induction hypothesis we have $\hat{S}(\mathcal{B}_i) = \sup\{\hat{J}(\mathcal{B}_i) \mid J \in X\}$ for $i = 1, \dots, n$; thus

$$\hat{S}(\mathcal{B}) = \dot{@}\left(\sup\{\hat{J}(\mathcal{B}_1) \mid J \in X\}, \dots, \sup\{\hat{J}(\mathcal{B}_n) \mid J \in X\}\right)$$

Note that, as X is a directed set so is each $\{\hat{J}(\mathcal{B}_i) \mid J \in X\}$ and, by the continuity of the aggregation operators, we have that

$$\begin{aligned}\hat{S}(\mathcal{B}) &= \sup\{\dot{@}(\hat{J}(\mathcal{B}_1),\dots,\hat{J}(\mathcal{B}_n)) \mid J \in X\}\\ &= \sup\{\hat{J}(\dot{@}(\mathcal{B}_1,\dots,\mathcal{B}_n)) \mid J \in X\}\\ &= \sup\{\hat{J}(\mathcal{B}) \mid J \in X\}\end{aligned}$$

Now we can consider the general case of a non-ground $\mathcal{B}$. In this case we have that, for all ground instantiation $\mathcal{B}\xi$ we have

$$\hat{S}(\mathcal{B}\xi) = \sup\{\hat{J}(\mathcal{B}\xi) \mid J \in X\}$$

and calculating infima we obtain the desired equality.

Theorem 13

(1) If all the operators occurring in the bodies of the rules of a program $\mathbb{P}$ are continuous, and the adjoint conjunctions are continuous in their second argument, then $T_{\mathbb{P}}$ is continuous.

(2) Conversely, if $T_{\mathbb{P}}$ is continuous for all program $\mathbb{P}$, then all the operators occurring in the bodies of the rules are continuous, and the adjoint conjunctions are continuous in their second argument

PROOF. (1). We have to check that for each directed subset of interpretations X and each ground atomic formula A

$$T_{\mathbb{P}}(\sup X)(A) = \sup\{T_{\mathbb{P}}(J)(A) \mid J \in X\}$$

Let us write $S = \sup X$, and consider the following chain of equalities:

$$\begin{aligned}T_{\mathbb{P}}(\sup X)(A) &= \sup\{\vartheta \mathbin{\dot{\&}_i} \hat{S}(\mathcal{B}\theta) \mid \langle C \leftarrow_i \mathcal{B}, \vartheta\rangle \in \mathbb{P}, A = C\theta\}\\ &\overset{(*)}{=} \sup\{\vartheta \mathbin{\dot{\&}_i} \sup\{\hat{J}(\mathcal{B}\theta) \mid J \in X\} \mid \langle C \leftarrow_i \mathcal{B}, \vartheta\rangle \in \mathbb{P}, A = C\theta\}\\ &\overset{(\star)}{=} \sup\{\vartheta \mathbin{\dot{\&}_i} \hat{J}(\mathcal{B}\theta) \mid J \in X, \text{ and } \langle C \leftarrow_i \mathcal{B}, \vartheta\rangle \in \mathbb{P}, A = C\theta\}\\ &= \sup\{\sup\{\vartheta \mathbin{\dot{\&}_i} \hat{J}(\mathcal{B}\theta) \mid \langle C \leftarrow_i \mathcal{B}, \vartheta\rangle \in \mathbb{P}, A = C\theta\} \mid J \in X\}\\ &= \sup\{T_{\mathbb{P}}(J)(A) \mid J \in X\}\end{aligned}$$

where equality $(*)$ follows from Lemma 12 and equality $(\star)$ follows from the continuity of the operators $\dot{\&}_i$.

The proof of item (2) is similar.

4 Procedural semantics

Once we know that the least model can be reached in at most countably many iterations, that is as ${T_{\mathbb{P}}}^{\omega}(\triangle)$, under pretty general assumptions, it is worth to define a procedural semantics which allows us to actually construct the answer to a query for a given program.

In this section we provide a procedural semantics to the paradigm of multi-adjoint logic programming, and a quasi-completeness theorem will be given.

In the following, we will be working in a hybrid language $\mathfrak{F}^e$ consisting of body formulas built up from elements of the lattice and propositional symbols; this way we can develop a symbolic computational model with partially evaluated formulas. The formal definition of the extended language is given below:

Definition 14 *Let $\mathbb{P}$ be a multi-adjoint logic program on a multi-adjoint algebra $\mathfrak{L}$ with carrier L and let V be the set of truth values of the rules in $\mathbb{P}$. The* extended language *$\mathfrak{F}^e$ is the corresponding algebra of body formulas freely generated from the disjoint union of V and the set of propositional symbols Π.*

Formulas in the language $\mathfrak{F}^e$ are called *extended formulas,* or simply *e-formulas.* An operator symbol ω interpreted under $\mathfrak{F}^e$ will be denoted as $\bar{\omega}$.

Our computational model will take a query (an atom), providing a lower bound of the value of A under any model of the program. Intuitively, the computation proceeds by, somehow, substituting atoms by lower bounds of their truth-value until, eventually, an extended formula with no atom is obtained, which will be interpreted in the multi-adjoint lattice to get the computed answer.

Given a program $\mathbb{P}$, we define the following admissible rules for transforming any pair formed by an e-formula and a substitution.

Definition 15 Admissible rules *for a pair (F, θ) where F is an e-formula and θ is a substitution, and A is an atom occurring in F (denoted $F[A]$), are the following:*

R1 Substitute $F[A]$ by $(F[A/\vartheta \mathbin{\bar{\&}_i} \mathcal{B}])\theta'$, and θ by $\theta' \circ \theta$ whenever
  (a) θ' is the mgu of C and A,
  (b) there exists a rule $\langle C \leftarrow_i \mathcal{B}, \vartheta\rangle$ in $\mathbb{P}$,

R2 Substitute A by $\bot$ (just to cope with unsuccessful branches), and do not modify θ.

R3 Substitute $F[A]$ by $(F[A/\vartheta])\theta'$ and θ by $\theta' \circ \theta$ whenever
  (a) θ' is the mgu of C and A
  (b) there exists a fact $\langle C \leftarrow_i \top, \vartheta\rangle$ in $\mathbb{P}$.

Note that if an e-formula turns out to have no atoms, then it can be directly interpreted in the multi-adjoint lattice $\mathcal{L}$. This justifies the following definition of *computed answer.*

Definition 16 *Let $\mathbb{P}$ be a program in a multi-adjoint language interpreted on a multi-adjoint lattice $\mathcal{L}$ and let $?A$ be a goal. An element $(\dot{@}[r_1, \dots, r_m], \theta)$, with $r_j \in L$, for all $j = 1, \dots, m$ is said to be a* computed answer *if there is a sequence $G_0, \dots, G_{n+1}$ such that*

(1) $G_0 = (A, id)$ and $G_{n+1} = (\bar{@}[r_1, \dots, r_m], \theta')$ where $\theta = \theta'$ restricted to the variables of A and $r_j \in L$ for all $j = 1, \dots m$.
(2) Every G_i, for $i = 1, \dots, n$, is a pair of an e-formula and a substitution.
(3) Every G_{i+1} is inferred from G_i by one of the admissible rules.

The length *of this computed answer is defined to be n.*

Note that the properties of multi-adjoint lattice guarantee that every computed answer is correct, therefore the correctness theorem for this procedural semantics is immediate.

4.1 Reductants

It might happen that for some lattices it is not possible to get the greatest correct answer, an example can be easily constructed for a non-linear lattice $\langle L, \preceq \rangle$ as follows: let a, b be two incomparable elements in L, and consider the situation in which a given query can be only matched with two rules, the first one leading to the answer a, and the second one leading to the answer b. By correctness of the procedural semantics we have that both a and b are correct answer and, therefore, it is obvious that the supremum of a and b is also a correct answer but uncomputable.

The idea to cope with this problem is the generalization of the concept of reductant. Namely, whenever we have k rules $\langle A \leftarrow_i @_i(D_1^i, \dots, D_{n_i}^i), \vartheta_i \rangle$ for $i = 1, \dots, k$, then there should exist a rule allowing us to get the greatest possible value of A under the program, that is, we would like to have the possibility of reaching the supremum of all the contributions *in a single step* of the computational model.

The *reductant property* is defined in [13] so that a single rule in the program computes the supremum stated above (which is a generalization of the concept of reductant [8]). Any rule $\langle A \leftarrow_i @_i(D_1^i, \dots, D_{n_i}^i), \vartheta_i \rangle$ contributes, by the adjoint property, with a value of the form $\vartheta_i \mathbin{\dot{\&}_i} b_i$ for the calculation of the lower bound for the truth-value of A; this fact justifies the definition below:

Definition 17 *Let $\mathbb{P}$ be a program, and A a ground atom, let $\langle C_i \leftarrow_{j_i} \mathcal{B}_i, \vartheta_i \rangle$ be the set of rules in $\mathbb{P}$ whose head matches with A (there are θ_i such that $A = C_i\theta_i$) and assume that this set contains at least a proper rule; a* reductant for *A is any rule $\langle A \leftarrow @(\mathcal{B}_1, \dots, \mathcal{B}_n)\theta, \top \rangle$ where*[4] *$\theta = \theta_1 \cdots \theta_n$, and $\leftarrow$ is any implication with an adjoint conjunctor, and the aggregator @ is defined as*

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

If the only rules matching with A turn out to be facts C_i, then the reductant is defined to be a fact which aggregates all the knowledge about A, i.e.,

$$\langle A \leftarrow \top, \sup\{\vartheta_1, \dots, \vartheta_n\} \rangle$$

It is immediate to prove that the rule constructed in the definition above, in presence of proper rules, behaves as a classical reductant for A in $\mathbb{P}$. Note that, as a consequence, the choice of the adjoint pair to represent the corresponding reductant is irrelevant for the computational model. Therefore, in the following, we will assume that our language has a distinguished adjoint pair to be selected in the construction of reductants, leading to the so-called *canonical reductants.*

Note that we have chosen to discard non-determinism by means of the use of reductants, following traditional techniques of logic programming in the rest of the construction.

In the following, we will assume that our program $\mathbb{P}$ has the reductant property, as a consequence we can also assume that the program contains all the reductants, since it can be easily proved that the set of models is not modified.

As a first consequence of the assumption of the reductant property, we present a result which states that any computed answer with length n can be improved by n iterations of the $T_{\mathbb{P}}$ operator, which also constructs computed answers as we will show later.

Proposition 18 *If $(\lambda, \theta_1 \cdots \theta_n)$ is a computed answer of length n for a ground goal $?A$, then $\lambda \preceq T^n_{\mathbb{P}}(\triangle)(A)$.*

PROOF. By induction on the length of the computed answer for $?A$.

[4] Note that the order is not important since the rules can be assumed to be standardized apart.

For $n = 1$ either R2 or R3 has been applied. The case of R2 is trivial, since $\lambda = \bot$; the case of R3 implies that there is a fact $\langle C \leftarrow \top, \vartheta\rangle$ and substitution θ such that $A = C\theta$, therefore the computed answer has the form (ϑ, θ) and:

$$\begin{aligned}\vartheta = \vartheta \mathbin{\dot{\&}_i} \top &\preceq \sup\{\vartheta \mathbin{\dot{\&}_i} T^0_{\mathbb{P}}(\triangle)(\mathcal{B}\theta) \mid \langle C \leftarrow_i \mathcal{B}, \vartheta\rangle \in \mathbb{P} \text{ and } A = C\theta\} \\ &= T^1_{\mathbb{P}}(\triangle)(A)\end{aligned}$$

Assume that the result is true for length less than n, and that $(\lambda, \theta_1 \cdots \theta_n)$ is a computed answer for $?A$ of length n, for $n > 0$. Obviously, the first step has been to apply R1 on a rule, say $\langle C \leftarrow_i \mathcal{B}, \vartheta\rangle$ with matching substitution θ_1, therefore we can assume that $G_1 = \vartheta \mathbin{\overline{\&}_i} \mathcal{B}\theta_1$.

Note that the formula $\mathcal{B}\theta_1$ is ground; and let us denote it by $@[B_1, \dots, B_l]$, where B_i are its ground atoms. Now, obviously, each atom B_i has a computed answer of length at most $n - 1$, which can be written as $(\lambda_i, \theta_2 \cdots \theta_n)$ for $i = 1, \dots, l$. By induction hypothesis, and the monotonicity of the $T_{\mathbb{P}}$ operator we have

$$\lambda_i \preceq T^{n-1}_{\mathbb{P}}(\triangle)(B_i) \text{ for all } i = 1, \dots, l$$

therefore

$$\begin{aligned}\lambda = \vartheta \mathbin{\dot{\&}_i} \dot{@}(\lambda_1, \dots, \lambda_l) &\preceq \vartheta \mathbin{\dot{\&}_i} \dot{@}(T^{n-1}_{\mathbb{P}}(\triangle)(B_1), \dots, T^{n-1}_{\mathbb{P}}(\triangle)(B_l)) \\ &= \vartheta \mathbin{\dot{\&}_i} \widehat{T^{n-1}_{\mathbb{P}}(\triangle)}(@[B_1, \dots, B_l]) \\ &= \vartheta \mathbin{\dot{\&}_i} \widehat{T^{n-1}_{\mathbb{P}}(\triangle)}(\mathcal{B}\theta_1) \\ &\preceq \sup\{\vartheta \mathbin{\dot{\&}_i} \widehat{T^{n}_{\mathbb{P}}(\triangle)}(\mathcal{B}\theta) \mid \langle C \leftarrow_i \mathcal{B}, \vartheta\rangle \in \mathbb{P}, A = C\theta\} \\ &= T^n_{\mathbb{P}}(\triangle)(A)\end{aligned}$$

4.2 Completeness result

The proof of the completeness result follows from the following proposition showing the behavior of both computed answers, which says that the $T_{\mathbb{P}}$ operator actually builds computed answers for ground goals. Specifically,

Proposition 19 *Let $\mathbb{P}$ be a program and A be a ground atom, then the pair $(T^n_{\mathbb{P}}(\triangle)(A), id)$ is a computed answer for $?A$ for all n.*

PROOF. By induction on n.

For $n = 0$, since $T^0_{\mathbb{P}}(\triangle)(A) = \bot$, the result follows as an application of R2.

As induction hypothesis, assume that $(T_{\mathbb{P}}^{n}(\triangle)(Q), id)$ is a computed answer for all ground goal $?Q$, and let us prove that $(T_{\mathbb{P}}^{n+1}(\triangle)(A), id)$ is also a computed answer for a given ground atom A. As we will use canonical reductants for A, we have to consider two possibilities attending to its two possible forms.

On the one hand, if the reductant is a fact, then all the rules matching with A are facts, and it should have the form $\langle A \leftarrow \top, \vartheta\rangle$, where ϑ is the supremum of the confidence values of the facts matching with A.

Now, by definition of $T_{\mathbb{P}}$ we have:

$$\begin{aligned} T_{\mathbb{P}}^{n+1}(\triangle)(A) &= \sup\{\vartheta_i \mathbin{\dot{\&}_{j_i}} \widehat{T_{\mathbb{P}}^{n}(\triangle)}(\mathcal{B}\theta) \mid \langle C \leftarrow_{j_i} \mathcal{B}, \vartheta_i\rangle \in \mathbb{P}, A = C\theta\} \\ &\overset{(*)}{=} \sup\{\vartheta_i \mid \langle C \leftarrow_{j_i} \top, \vartheta_i\rangle \in \mathbb{P}, A = C\theta\} = \vartheta \end{aligned}$$

where the equality $(*)$ holds because no proper rule (only facts) in $\mathbb{P}$ match with A.

By using the canonical reductant, the sequence

$$G_0 = (A, id) \qquad G_1 = (\vartheta, id)$$

proves that (ϑ, id) is a computed answer for $?A$.

On the other hand, if $\langle A \leftarrow @(\mathcal{B}_1, \dots, \mathcal{B}_l)\theta, \top\rangle$ is a ground instance of the canonical reductant for A, the body of the reductant can be interpreted as a composition of a number of aggregators depending on a number of atoms B_i, which will be denoted as $@'[B_1, \dots, B_k]$. Thus we have:

$$\begin{aligned} T_{\mathbb{P}}^{n+1}(\triangle)(A) &= \sup\{\vartheta \mathbin{\dot{\&}_i} \widehat{T_{\mathbb{P}}^{n}(\triangle)}(\mathcal{B}\phi) \mid \langle C \leftarrow_i \mathcal{B}, \vartheta\rangle \in \mathbb{P}, A = C\phi\} \\ &\overset{(*)}{=} \dot{@}(\widehat{T_{\mathbb{P}}^{n}(\triangle)}(\mathcal{B}_1\theta), \dots, \widehat{T_{\mathbb{P}}^{n}(\triangle)}(\mathcal{B}_l\theta)) \\ &= \widehat{T_{\mathbb{P}}^{n}(\triangle)}(@(\mathcal{B}_1, \dots, \mathcal{B}_l)\theta) \\ &= \widehat{T_{\mathbb{P}}^{n}(\triangle)}(@'[B_1, \dots, B_k]) \\ &= \dot{@}'(T_{\mathbb{P}}^{n}(\triangle)(B_1), \dots, T_{\mathbb{P}}^{n}(\triangle)(B_k)) \end{aligned}$$

where $(*)$ follows by the definition of reductant.

By induction hypothesis, $(T_{\mathbb{P}}^{n}(\triangle)(B_i), id)$ is a computed answer for $?B_i$ for all ground atom B_i, with computing sequence

$$H_{i_0} = (B_i, id), \dots, H_{i_j} = (T_{\mathbb{P}}^{n}(\triangle)(B_i), id)$$

Consider $G_0 = (A, id)$, and generate G_1 as an application of the canonical reductant for A, that is $G_1 = (\top \mathbin{\bar{\&}} \bar{@}'[B_1, \dots, B_k], id)$. Now, we can use the computing sequences for B_i to obtaining[5]

$$G_r = (\top \mathbin{\bar{\&}} \bar{@}'[T^n_{\mathbb{P}}(\triangle)(B_1), \dots, T^n_{\mathbb{P}}(\triangle)(B_k)], id)$$

The proof of the completeness theorem follows easily from the previous theorem.

Theorem 20 (Approximative-completeness) *Given a program $\mathbb{P}$, for every correct answer $(\lambda; \theta)$ for a program $\mathbb{P}$ and a ground goal $?A$, there is a sequence of computed answers (λ_n, id) such that $\lambda \preceq \sup\{\lambda_n \mid n \in \mathbb{N}\}$.*

PROOF. By Proposition 19 we have that, for all n, $(T^n_{\mathbb{P}}(\triangle)(A), id)$ is a computed answer for $?A$.

Now, consider $\lambda_n = T^n_{\mathbb{P}}(\triangle)(A)$; as $(\lambda; \theta)$ is a correct answer, and $T^\omega_{\mathbb{P}}(\triangle)$ is, in particular, a model, we have that

$$\begin{aligned}
\lambda &\preceq \inf_{\xi} \left\{ T^\omega_{\mathbb{P}}(\triangle)(A\theta\xi) \right\} \\
&= T^\omega_{\mathbb{P}}(\triangle)(A) = \sup\{T^n_{\mathbb{P}}(\triangle)(A) \mid n \in \mathbb{N}\} = \sup\{\lambda_n \mid n \in \mathbb{N}\}
\end{aligned}$$

This approximative completeness theorem cannot be extended to a full completeness theorem. It is not difficult to show the existence of programs for which the greatest correct answer for a query $?A$, that is $T_{\mathbb{P}}{}^{\omega}(\triangle)(A)$, can be approximated up to any value, but not attainable in finitely many steps.

Regarding non-ground queries, all the results in this section can be obtained by means of suitable versions of lifting lemmas; just consider that we are using classical crisp unification in our computational model. Moreover, in the next section we will show how a similarity-based unification approach can be constructed on it.

5 Similarity-based unification

Our approach will consider similarities acting on elements of domains of attributes. This was already motivated in the introduction, when we have heterogeneous data source, and aiming to solve problems for the semantic web

[5] This is due to the fact that the B_i are ground.

we would have to link, e.g. the relation $human(name, birth\_date, address)$ to the relation $person(name, birth\_date, address)$. The idea is based on the fact that part of our knowledge base, the multi-adjoint program $\mathbb{P}$, might consist of graded facts representing information about existent similarities on different domains which depend on the predicate they are used in, e.g.

$$\langle s(\text{Salaš}, \text{Szałas}), 0.8\rangle \qquad \langle s(\text{Salaš}, \text{Szálás}), 0.7\rangle \qquad \langle s(\text{Szálás}, \text{Szałas}), 0.9\rangle$$

The particular semantics of our logic, based on the multi-adjoint paradigm, enables one to easily implement a version of fuzzy unification by extending suitably our given program.

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

$$\langle s(x,x), \top\rangle \qquad \langle s(x,y) \leftarrow s(y,x), \top\rangle \qquad \langle s(x,z) \leftarrow s(x,y) \mathbin{\&} s(y,z), \top\rangle$$

For all function symbol we also have

$$\langle s(f(x_1,\dots,x_n), f(y_1,\dots,y_n)) \leftarrow s^f_1(x_1,y_1) \mathbin{\&} \cdots \mathbin{\&} s^f_n(x_n,y_n), \top\rangle$$

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

$$\langle P(y_1,\dots,y_n) \leftarrow P(x_1,\dots,x_n) \mathbin{\&} s^P_1(x_1,y_1) \mathbin{\&} \cdots \mathbin{\&} s^P_n(x_n,y_n), \top\rangle$$

where & is some conjunction suitably describing the situation formalized by the program. Note that the implication in these axioms can be arbitrary due to both, the adjoint property and the fact that the truth-value of each rule is always $\top$.

This way we get a multi-adjoint logic program $\mathbb{P}_E$ in which it is possible to get computed answers wrt $\mathbb{P}_E$ with similarity match in unification. This justifies the following extension of Definition 16:

Definition 21 *Let $\mathbb{P}$ be a program, let $\mathbb{P}_E$ be an extension of this program by appropriate rules describing axioms of similarity for the respective predicate and function symbols, and let $?A$ be a query. An element $(\dot{@}[r_1,\dots,r_m],\theta)$, with $r_j \in L$, for all $j = 1,\dots,m$ is said to be a* similarity-based computed answer *for $?A$ wrt the program $\mathbb{P}$ if it is a computed answer for $?A$ wrt $\mathbb{P}_E$.*

Note that similarity-based computed answers are nothing but computed answers with crisp unification on a program extended by axioms of equality.

Moreover, as this approach makes use of the computational model of multi-adjoint programs, our similarity-based unification model, could also get benefit from existing efficient implementations of its operational semantics [11].

Comparison with other approaches

A lot of research has been done in the field of unification based on similarities, for instance, [1] starts from unification based on similarity to derive a logic programming system, thoroughly relying on similarity, one of its principal features being the allowance of flexible information retrieval in deductive data bases. On the other hand, the purpose of [16] is to investigate the use of similarities as the basis for unification and resolution in logic programming. As a result, a well-founded semantical method to incorporate linguistic variables into logic programming is given.

In contrast, the approach taken in [6] is based on fuzzy set theory and computing something like the degree of being a singleton. They are restricted to max-min connectives and we argue that similarity should be coupled by product semantics, at least when working in the unit interval. The reason is that the truth of the query answer should be independent of the fact whether we made a misprint and/or a translation error or not (one of possible reasons for having no unification but similarity one).

An specially interesting connection can be established between our approach and the weak unification algorithm introduced in [15]. With this aim, in the rest of the section we will work in a particular case of the multi-adjoint framework in which $L = [0, 1]$ and the only connectives will be Gödel's implication and Gödel's conjunction.

We recall in the following paragraphs, in order to make this paper as self-contained as possible, Sessa's *weak unification algorithm*, which aims at finding a weak mgu of two atoms A and B starting from a set of equations W associated to A and B.

> A weak unifier for $A = P(t_1, \dots, t_n)$ and $B = Q(t'_1, \dots, t'_n)$ with unification degree λ up to a similarity relation $\mathcal{R}$ *(or, simply, a* λ-unifier*) is a substitution* θ *such that*
>
> $$\lambda = \min_{1\le i\le n} \left\{\mathcal{R}(P,Q), \mathcal{R}(t_i\theta, t'_i\theta)\right\} = \max_{\vartheta\in\Psi} \left\{ \min_{1\le i\le n} \{\mathcal{R}(P,Q), \mathcal{R}(t_i\vartheta, t'_i\vartheta)\}\right\}$$
>
> *where* Ψ *denotes the set of all the substitutions.*
>
> *The procedure constructs a sequence of* improved *sets of equations until a solved set is reached. The algorithm is given below:*

Given two atoms $A = P(t_1, \dots, t_n)$ and $B = Q(t'_1, \dots, t'_n)$ of the same arity with no common variables to be unified, construct its associated set of equations W. If $\mathcal{R}(P, Q) = 0$, halts with failure, otherwise, set $\lambda = \mathcal{R}(P, Q)$ and $W = W \setminus \{P = Q\}$.

Until the current set of equation W does not change, non-deterministically choose from W an equation of a form below and perform the associated action.

(1) $f(t_1, \dots, t_n) = g(t'_1, \dots, t'_n)$ where $\mathcal{R}(f, g) > 0$ replace by the equations $t_1 = t'_1, \dots, t_n = t'_n$ and set $\lambda = \min\{\lambda, \mathcal{R}(f, g)\}$.
(2) $f(t_1, \dots, t_n) = g(t'_1, \dots, t'_n)$ where $\mathcal{R}(f, g) = 0$; halts with failure.
(3) $x = x$ delete the equation.
(4) $t = x$ where t is not a variable; replace by the equation $x = t$.
(5) $x = t$ where $x \neq t$ and x has another occurrence in the set of equations: if x appears in t then halt with failure, otherwise perform the substitution $\{x/t\}$ in every other equation.

Similarities are considered, in [15], to act on constants, function symbols and predicate symbols, therefore, assume that we have a similarity relation $\mathcal{R}$ acting on $\mathcal{F} \cup \mathcal{P} \cup \mathcal{C}$ (function, predicate and constant symbols) with truth-values ranging in the unit real interval $[0, 1]$. In our approach, the similarity $\mathcal{R}$ is internalized, that is, we include a new predicate symbol in our language, denoted $s_{\mathcal{R}}$.

Now, for every $f, g \in \mathcal{F}$ with $\mathcal{R}(f, g) > 0$ let us extend our logic program by the following schema of axioms

$$\langle s_{\mathcal{R}}(f(x_1, \dots, x_n), g(y_1, \dots, y_n)) \leftarrow s_{\mathcal{R}}(x_1, y_1) \ \& \ \cdots \ \& \ s_{\mathcal{R}}(x_n, y_n), \mathcal{R}(f, g)\rangle$$

where, in the case of constants (as 0-ary functions) it is understood as

$$\langle s_{\mathcal{R}}(c, d), \mathcal{R}(c, d)\rangle$$

In addition, for every $P, Q \in \mathcal{P}$ with $\mathcal{R}(P, Q) > 0$ let us extend our logic program by a schema of axioms

$$\langle P(y_1, \dots, y_n) \leftarrow Q(x_1, \dots, x_n) \,\&\, s_{\mathcal{R}}(x_1, y_1) \,\&\, \cdots \,\&\, s_{\mathcal{R}}(x_n, y_n), \mathcal{R}(P, Q)\rangle$$

The extension of a program $\mathbb{P}$ with respect to a similarity relation $\mathcal{R}$ described above will be denoted $\mathbb{P}_{\mathcal{R}}$.

It is worth to mention that the equality axioms introduced in order to obtain $\mathbb{P}_E$ are simply the similarity-based extension obtained when the 'external' similarity $\mathcal{R}$ is the usual equality relation.

In the following, in order to emulate the weak unification algorithm, we will be working with the two-fold extension of the program $\mathbb{P}_{E\mathcal{R}}$. Moreover, in this particular case, we will not need to have any other rules than those in the two-fold extension of an empty program $\varnothing_{E\mathcal{R}}$, denoted $E \cup \mathcal{R}$.

Theorem 22 *Let $P(t_1, \dots, t_n)$ and $Q(t'_1, \dots, t'_n)$ be two atoms, assume that some substitution θ is a λ-unifier (for $\lambda \in [0,1]$) obtained by the weak unification algorithm, then (λ, θ) is a computed answer to the query $?P(t_1, \dots, t_n)$ wrt the program $E \cup \mathcal{R} \cup \{\langle Q(t'_1, \dots, t'_n), 1\rangle\}$.*

PROOF. As the weak unification algorithm is non-deterministic, we will withdraw the use of reductants from our calculation, so that our computational model turns back to be non-deterministic.

By induction along a branch of the computation of the weak unification algorithm. We will show how the multi-adjoint computational model emulates each possible step in the weak unification algorithm:

Preprocessing step. Set $\lambda = \mathcal{R}(P, Q)$. The emulation makes use of the rule

$$\langle P(y_1, \dots, y_n) \leftarrow Q(x_1, \dots, x_n) \,\&\, s_{\mathcal{R}}(x_1, y_1) \,\&\, \cdots \,\&\, s_{\mathcal{R}}(x_n, y_n), \mathcal{R}(P, Q)\rangle$$

which builds the following e-formula

$$\min\Big\{\mathcal{R}(P, Q), Q(x_1, \dots, x_n) \,\&\, s_{\mathcal{R}}(x_1, t_1) \,\&\, \cdots \,\&\, s_{\mathcal{R}}(x_n, t_n)\Big\}$$

now, by using the fact $\langle Q(t'_1, \dots, t'_n), 1\rangle$, we obtain

$$\min\Big\{\mathcal{R}(P, Q), s_{\mathcal{R}}(t'_1, t_1) \,\&\, \cdots \,\&\, s_{\mathcal{R}}(t'_n, t_n)\Big\}$$

which represents both the set of equations W of the weak unification algorithm and the current value of the unification degree.

We will show below our emulation of the applications of Steps 1–5 in the weak unification algorithm.

Step 1. *Given $t_1 = f(u_1, \dots, u_m) = g(v_1, \dots, v_m) = t'_1$, with $\mathcal{R}(f, g) > 0$, replace by the equalities $u_1 = v_1, \dots, u_m = v_m$ and update $\lambda = \min(\lambda, \mathcal{R}(f, g))$.*

The computation uses the axiom

$$\begin{aligned}\langle s_{\mathcal{R}}(f(x_1, \dots, x_m), g(y_1, \dots, y_m)) \leftarrow& \\ & s_{\mathcal{R}}(x_1, y_1) \;\&\; \cdots \;\&\; s_{\mathcal{R}}(x_m, y_m), \mathcal{R}(f, g)\rangle\end{aligned}$$

which builds the new e-formula below

$$\min\Big\{\mathcal{R}(f,g),\mathcal{R}(P,Q),s_{\mathcal{R}}(u_1,v_1)\ \&\ \cdots\ \&\ s_{\mathcal{R}}(u_m,v_m),\\ s_{\mathcal{R}}(t'_2,t_2)\ \&\ \cdots\ \&\ s_{\mathcal{R}}(t'_n,t_n)\Big\}$$

Step 2. *If $f(t_1,\dots,t_n)=g(t'_1,\dots,t'_n)$ where $\mathcal{R}(f,g)=0$; then halt with failure.*

This step is never applied because we are assuming that θ is a λ-unifier, therefore the algorithm does not terminate with failure.

Step 3. *If $x=x$ delete the equation.*

As we are using crisp unification, this step is trivial.

Step 4. *If $t=x$ where t is not a variable; replace by the equation $x=t$.*

There is nothing to simulate here, since all the equations are interpreted as similarities (which turn out to be commutative).

Step 5. *If $x=t$ where $x\neq t$ and x has another occurrence in the set of equations: if x appears in t then halt with failure, otherwise perform the substitution $\{x/t\}$ in every other equation.*

Once again, we are assuming that the algorithm does not halt with failure. The application of the substitution is part of our semantics (see Definition 15), because we use the crisp model of unification.

Recall that, when working with max-min similarities it is possible to decompose the similarity to a sequence of refining crisp equivalences. This is no longer possible in our case, due to the greater generality of the multi-adjoint approach. Moreover, in [15], the similarity-based SLD derivation is applied to a classical program (for there is no uncertainty in the program). The only place where uncertainty appears is in the similarity coming from unification. Now, as a consequence of the theorem on emulation of unification by the computational model of multi-adjoint programs, we can obtain the following theorem, in which we are assuming the language of [15] (Defn.7.2).

Theorem 23 *Given a similarity $\mathcal{R}$, a crisp program $\mathbb{P}$ and a goal G_0, and a similarity-based derivation*

$$G_0 \Longrightarrow_{C_1,\theta_1,\lambda_1} G_1 \Longrightarrow \cdots \Longrightarrow_{C_m,\theta_m,\lambda_m} G_m$$

the approximation degree of $\theta_1\cdots\theta_n$ restricted to the variables of G_0 is set to *be* $\lambda = \min_{1 \le i \le m}\{\lambda_i\}$*, then there exists a multi-adjoint computation for* $?G_0$ *and the (crisp) program* $\mathbb{P}$ *in the logic with Gödel connectives and* $L = [0,1]$ *such that the computed answer is* (λ, θ)*.*

It is worth to note that all theorems on fix-point, H_λ and $\mathcal{P}_\lambda$ semantics given in [15] state that Sessa's approach perfectly embeds in our more general multi-adjoint approach (suitable restricted to the unit interval and Gödel connectives).

6 Conclusions

A general framework of logic programming which allows the simultaneous use of different implications in the rules and rather general connectives in the bodies have been introduced. A procedural semantics for this framework of multi-adjoint logic programming has been presented, and a quasi-completeness theorem proved. On this computational model, a similarity-based unification approach is constructed by simply adding axioms of fuzzy similarities and using classical crisp unification which provides a semantic framework for logic programming with different notions of similarity.

In the final section, our approach to unification is compared with some other approaches, we show that the weak unification algorithm introduced in [15] can be emulated by our unification model. From a practical point of view, the proposed approach seems to be appropriate for some applications for information retrieval systems such as those studied in [10].

References

[1] F. Arcelli and F. Formato. Likelog: a logic programming language for flexible data retrieval. In *ACM Symposium on Applied Computing.*, pages 260–267, 1999.

[2] C.V. Damásio and L. Moniz Pereira. Monotonic and residuated logic programs. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU'01*, pages 748–759. Lect. Notes in Artificial Intelligence, 2143, 2001.

[3] S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, and I. Horrocks. The semantic web: the roles of XML and RDF. *IEEE Internet Computing*, 43:2–13, 2000.

[4] R. P. Dilworth and M. Ward. Residuated lattices. *Transactions of the American Mathematical Society*, 45:335–354, 1939.

[5] R. Fagin and E. L. Wimmers. A formula for incorporating weights into scoring rules. *Theoretical Computer Science*, 239:309–338, 2000.

[6] F. Formato, G. Gerla, and M.I. Sessa. Similarity-based unification. *Fundamenta Informaticae*, 41(4):393–414, 2000.

[7] K.G. Jeffery. What's next in database. *ERCIM News*, 39:24–26, 1999.

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

[9] P. Kriško, P. Marcinšák, P. Mihók, J. Sabol, and P. Vojtáš. Low retrieval remote querying dialogue with fuzzy conceptual, syntactical and linguistical unification. In *Flexible Query Answering Systems, FQAS'98*, pages 215–226. Lect. Notes in Comp. Sci. 1495, 1998.

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

[11] J. Medina, E. Mérida-Casermeiro, and M. Ojeda-Aciego. Multi-adjoint logic programming: a neural net approach. In *Logic Programming. ICLP'02*. Lect. Notes in Computer Science, 2002. To appear.

[12] J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Multi-adjoint logic programming with continuous semantics. In *Logic Programming and Non-Monotonic Reasoning, LPNMR'01*, pages 351–364. Lect. Notes in Artificial Intelligence 2173, 2001.

[13] J. Medina, M. Ojeda-Aciego, and P. Vojtáš. A procedural semantics for multi-adjoint logic programming. In *Progress in Artificial Intelligence, EPIA'01*, pages 290–297. Lect. Notes in Artificial Intelligence 2258, 2001.

[14] J. Pavelka. On fuzzy logic I, II, III. *Zeitschr. f. Math. Logik und Grundl. der Math.*, 25, 1979.

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

[16] H. Virtanen. Łukasiewicz logic programming based on fuzzy equality. In *Intelligent Techniques and Soft Computing Proceedings, EUFIT'96*, pages 646–650, 1996.

[17] P. Vojtáš. Fuzzy logic programming. *Fuzzy sets and systems*, 124(3):361–370, 2001.

[18] P. Vojtáš and L. Paulík. Soundness and completeness of non-classical extended SLD-resolution. In *Extensions of Logic Programming, ELP'96*, pages 289–301. Lect. Notes in Comp. Sci. 1050, 1996.