Similarity-Based Unification: A Multi-Adjoint Approach

Jesús Medina, Manuel Ojeda-Aciego,
Dept. Matemática Aplicada.*
Universidad de Málaga.
{jmedina,aciego}@ctima.uma.es

Peter Vojtáš
Dept. Mathematical Informatics.†
P.J. Šafárik University.
vojtas@kosice.upjs.sk

Abstract

The aim of this paper is to build a formal model for fuzzy unification in multi-adjoint logic programs containing both a declarative and a procedural part, and prove its soundness and completeness. Our approach is based on a general framework for logic programming, which gives a formal model of fuzzy logic programming extended by fuzzy similarities and axioms of first-order logic with equality.

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 and satisfaction) and the procedural part, more focused on algorithmic aspect 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 (that is, identical) statements can be handled identically. In the classical case, on the one hand, there was no need to 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. 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.

In this paper we stress the fact that, for fuzzy unification, both a procedural and declarative semantics are needed, as opposed to the two valued case.

Multi-adjoint (MA) logic programming

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 [6, 7]. We recall definitions of declarative and procedural semantics of multi-adjoint logic programming and show how its soundness and completeness, and especially the fixpoint 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 intuition behind MA logic programs is as fol-

* Partially supported by Spanish DGI project BFM2000-1054-C02-02 and Junta de Andalucía project TIC-115.

† Partially supported by Slovak project VEGA 1/7557/20

lows: Considering different implication operators, such as Lukasiewicz, 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 [7] to introduce MA logic programs as an extension of monotonic logic programs, presented in [2, 3] so that it is possible to use a number of different implications in the rules of our programs.

The semantics of generalized logic programs requires a notion of consequence (implication) which satisfies a generalized modus ponens rule. It is natural to look for a semantic basis as a common denominator of the residuated lattices and fairly general conjunctors and their adjoints, as used in [9]. The main point in the extension to a more general setting, in which different implications (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* (semi)lattice.

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

($\ell 1$) $\langle L, \preceq \rangle$ is bounded, i.e. it has bottom ($\bot$) and top ($\top$) elements;

($\ell 2$) $(\leftarrow_i, \&_i)$ is an adjoint pair in $\langle L, \preceq \rangle$ for $i = 1, \ldots, n$;

($\ell 3$) $\top \,\&_i\, \vartheta = \vartheta \,\&_i\, \top = \vartheta$ for all $\vartheta \in L$ for $i = 1, \ldots, n$.

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

Syntax and Semantics of MA 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 language $\mathfrak{F}$. Notice that we are allowed to use different implications in our rules.

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

1. The *rule* $(A \leftarrow_i \mathcal{B})$ is a formula of $\mathfrak{F}$;
2. The *confidence factor* ϑ is an element (a truth-value) of L;
3. The *head* A is an atom;
4. The *body* $\mathcal{B}$ is a formula built from atoms $B_1, \ldots, B_n$ $(n \geq 0)$ by the use of conjunctors, disjunctors, and aggregators.
5. *Facts* are rules with body $\top$.
6. A *query* (or *goal*) is an atom intended as a question $?A$ prompting the system.

Definition: An *interpretation* is a mapping from the Herbrand base of the program to the multi-adjoint semilattice of truth-values L. The set of all interpretations of the formulas is denoted $\mathcal{I}_{\mathfrak{L}}$.

The ordering of the truth-values can be easily extended to the set of interpretations.

Definition: Given an interpretation $I \in \mathcal{I}_{\mathfrak{L}}$, a weighted rule $\langle A \leftarrow_i \mathcal{B}, \vartheta \rangle$ is *satisfied* by I iff $\vartheta \preceq \hat{I}(A \leftarrow_i \mathcal{B})$, where for a given formula F we have

$$\hat{I}(F) = \inf\{I(F\theta) \mid \theta \text{ is a ground substitution}\}$$

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

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

Definition: Let $\mathbb{P}$ be a multi-adjoint logic program. The *immediate consequences operator* $T^{\mathfrak{L}}_{\mathbb{P}}: \mathcal{I}_{\mathfrak{L}} \to \mathcal{I}_{\mathfrak{L}}$, mapping interpretations to interpretations, is defined by considering $T^{\mathfrak{L}}_{\mathbb{P}}(I)(A)$ as

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

As it is usual in the logic programming framework, the semantics of a multi-adjoint logic program is defined as the least-fixpoint of $T^{\mathfrak{L}}_{\mathbb{P}}$. In [7], the monotonicity of $T^{\mathfrak{L}}_{\mathbb{P}}$, 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 $T^{\mathfrak{L}}_{\mathbb{P}}$ can be extended to first-order multi-adjoint logic programs; therefore, the least fixpoint of $T^{\mathfrak{L}}_{\mathbb{P}}$ can be obtained by at most countably many iterations of $T^{\mathfrak{L}}_{\mathbb{P}}$ from the least interpretation.

Procedural semantics

Once we know that the least model can be reached in at most countably many iterations, it is worth to define a procedural semantics which allows us to actually construct the answer to a query for a given program.

In the following, we will be working in a hybrid language $\mathfrak{F}^e$ made up from the elements of the lattice, and the same alphabet of the language without the adjoint implicators.

We will refer to the formulas in the language $\mathfrak{F}^e$ simply as *extended formulas,* or *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), and will provide 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 semilattice 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: *Admissible rules* are the following, for a pair (F, θ) and an atom A occurring in a formula F, which we denote $F[A]$:

1. Substitute $F[A]$ by $(F[A/\vartheta \bar{\&}_i \mathcal{B}])\theta'$, 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 $\mathbb{P}$,
2. Substitute A by $\bot$ (just to cope with unsuccessful branches).
3. 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 semilattice L. This justifies the definition of *computed answer.*

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

1. $G_0 = (A, id)$ and $G_{n+1} = (\bar{@}[r_1, \ldots, r_m], \theta')$ where $\theta = \theta'$ restricted to the variables of A and $r_i \in L$ for all $i = 1, \ldots n$.
2. Every G_i, for $i = 1, \ldots, n$, 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.

Note that every computed answer is correct, this is guaranteed by the properties of multi-adjoint semilattice.

It might be the case that for some lattices it is not possible to get the correct answer, for instance when we have a finite number of rules $A \xleftarrow{\vartheta_i}_i @_i(D_1^i, \ldots, D_{n_i}^i)$ for $i = 1, \ldots, k$, and we are working with a non-linear lattice of truth-values.

As any rule $A \xleftarrow{\vartheta_i}_i @_i(D_1^i, \ldots, D_{n_i}^i)$ contributes with the value $\vartheta_i \,\dot{\&}_i\, b_i$ for the calculation of the lower bound for the truth-value of A, we would like to have the possibility of reaching the supremum of all the contributions, in the computational model, in a single step. The *reductant property* is defined in [6] so that a single rule in the program computes the supremum stated above (which is a generalisation of the concept of reductant [5]):

The proof of the completeness theorem follows from the fact that the least fix-point is also the least model of a program.

Theorem 1 *Assume a MA program $\mathbb{P}$ with the reductant property such that $T_{\mathbb{P}}^{\mathfrak{L}}$ is continuous. For every correct answer (λ, θ) for a program $\mathbb{P}$ and a query $?A$, there exists a chain of elements λ_n such that $\lambda \preceq \sup \lambda_n$, such that for arbitrary n_0 there exists a computed answer (δ, θ') such that $\lambda_{n_0} \preceq \delta$ and θ' is more general than θ.*

Similarity-based unification

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 [10] 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.

The particular semantics given to our logic, based on the multi-adjoint paradigm, enables one to easily implement a version of fuzzy unification. When extending arbitrary logic program by axioms of equality with truth-value $\top$

$$p(y_1,\ldots,y_n) \leftarrow_i p(x_1,\ldots,x_n)\&_P x_1 =_1 y_1 \ \&_P \cdots \&_P \ x_n =_n y_n$$

and axioms of similarity also with truth-value $\top$

$$\begin{aligned}
&s(x,x)\\
&s(x,y) \ \leftarrow_i \ s(y,x)\\
&s(x,z) \ \leftarrow_i \ s(x,y) \ \&_P \ s(y,z)\\
&p(y_1,\ldots,y_n) \leftarrow_i\\
&\quad p(x_1,\ldots,x_n) \ \&_P \ s_1(x_1,y_1) \ \&_P \ \cdots \ \&_P \ s_n(x_n,y_n)\\
&s(f(x_1,\ldots,x_n),f(y_1,\ldots,y_n)) \leftarrow_i\\
&\qquad\qquad s_1(x_1,y_1) \ \&_P \ \cdots \ \&_P \ s_n(x_n,y_n)
\end{aligned}$$

it is possible to get computed answers with similarity match in unification.

In contrast, the approach taken in [4] 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 has to 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).

Note that the implication in these similarity and equality axioms can be arbitrary due to both, property ($\ell 3$) of multi-adjoint semilattice and the fact that the truth-value of each rule is always $\top$.

Conclusions

We express properties of fuzzy similarities and axioms of first-order calculus with equality as additional rules of our MA program $\mathbb{P}$. Then the fixpoint theorem and minimal model iteration gives a model of this extended theory. So similarity based fuzzy unification is nothing else as what the $T_{\mathbb{P}}^{\mathcal{L}}$ operator does, closure of crisp unification under similarity and equality axioms. And since countably many iterations of $T_{\mathbb{P}}^{\mathcal{L}}$ is also a model of this extended theory, we have a sound and complete model of fuzzy unification.

References

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

[2] C.V. Damásio and L. Moniz Pereira. Hybrid probabilistic logic programs as residuated logic programs. In *Proc. JELIA'00*, pages 57–73. Lect. Notes in AI, 1919, Springer-Verlag, 2000.

[3] C.V. Damásio and L. Moniz Pereira. Monotonic and residuated logic programs. In *Proc. ECSQARU'01*. Lect. Notes in Comp. Sci., Springer-Verlag, 2001. To appear

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

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

[6] J. Medina, M. Ojeda-Aciego, and P. Vojtáš. A procedural semantics for multi-adjoint logic programming. Available at the following web address `www.satd.uma.es/aciego/TR/procsem-tr.pdf`.

[7] J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Multi-adjoint logic programming with continuous semantics. In *Proc. LPNMR'01*, 2001. To appear.

[8] P. Vojtáš. Declarative and procedural semantics of fuzzy similarity based unification. *Kybernetika*, 36(6):707–720, 2000.

[9] P. Vojtáš. Fuzzy logic programming. *Fuzzy sets and systems*, 2001. Accepted.

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