Extending Unification through Similarity Relations

Francesca Arcelli[1], Ferrante Formato[2] and Giangiacomo Gerla[3]

[1] DIIIE Università di Salerno, I-84084 Fisciano (SA), Italy.
email: arcelli@ponza.dia.unisa.it
[2] DIIMA Università di Salerno, I-84084 Fisciano (SA), Italy.
email: formato@ponza.dia.unisa.it
[3] DIIMA Università di Salerno, I-84084 Fisciano (SA), Italy.
email: gerla@matna2.dma.unina.it

1 Introduction

Unification is crucial in a wide variety of topics, ranging from artificial intelligence to automated theorem proving. In the field of approximated reasoning, an extended kind of unification assumes the same importance. A widely accepted and complete definition of fuzzy unification has not been proposed in the literature yet . Our approach to fuzzy unification lies on the definition of similarity relation. Several theoretical works have analyzed this relation and its properties. All these aspects are strictly connected to the specific problem domain. Given a first order language $\mathcal{L}$, with a set of predicate names $\mathcal{R}$ and a set of constants names $\mathcal{C}$, we define two similarity relations on $\mathcal{R}$ and $\mathcal{C}$. We expand the set of terms of the language by considering a constant as a "cloud": a cloud is a set of constants in $\mathcal{C}$ which can be seen as "undefinetely similar" each other; therefore we introduce a measure indicating the extent to which a cloud collapses into a singleton. This degree is used to expand the relation of similarity in $\mathcal{C}$ to a similarity relation on the generalized set of constants.

Starting from the first-order terms unification algorithm defined by Martelli and Montanari ([15]), we describe a fuzzy extension of this algorithm. The fuzzy unification algorithm is based on fuzzy similarity relations and the cloud are arised whenever a "constants mismatch" occurs. The extended unification algorithm takes a system of term equations as input and gives a system of multiequations as output. The output system is composed of "normalized" multiequations i.e. such that the lefthand side and righ-hand side are respectively disjoint. In a normalized system of term equations, each variable is bounded to at most a cloud, so an "expanded" unifier is immediately available. The singleton measure of the clouds in the normalized system can be seen as the "cost" one must pay in order to have a classical solution, i.e. a binding between a variable and a constant in $\mathcal{C}$.

2 Fuzzy Unification

The unification algorithm of standard logic programming is extended to the fuzzy context using similarities relations.

2.1 Similarity Relations

A similarity relation is a fuzzy relation that extends in a given multivalued logic the properties of the classical equivalence relation. Recall that an equivalence relation on a set S is a model of the system of axioms

$$r(x,x) = 1 \text{ (reflexivity)}$$
$$r(x,y) \rightarrow, r(y,x) \text{ (symmetry)}$$
$$r(x,y) \wedge r(y,z) \rightarrow r(x,z). \text{ (transitivity)}$$

Since a formula α is true in a multivalued interpretation I if $I(\alpha) = 1$, an implication $\alpha \Rightarrow \beta$ is true in I if $I(\beta) \geq I(\alpha)$ and $I(\alpha \wedge \beta) = I(\alpha) \wedge' I(\beta) = 1$, the axiomatic system for classical equivalence turns into the following definition of "similarity" relation:

Definition 1. A similarity relation on a domain $\mathcal{U}$ is a fuzzy subset $\mathcal{R}$ of $\mathcal{U} \times \mathcal{U}$ such that the following properties hold:

i) $\mathcal{R}(a,a) = 1$ for any $a \in \mathcal{U}$
ii) $\mathcal{R}(a,b) \geq \mathcal{R}(b,a)$ for any pair of elements $a, b \in \mathcal{U}$
iii) $\mathcal{R}(a,c) \geq \mathcal{R}(a,b) \wedge \mathcal{R}(b,c)$ for any $a, b, c \in \mathcal{U}$.

Given a domain $\mathcal{U}$ and fuzzy similarity relation $\mathcal{R}$ on $\mathcal{U}$, we can associate to every set of elements of $\mathcal{U}$ a degree expressing the extent to which the subset collapses into a singleton. Each set of elements in $\mathcal{U}$ can be seen as composed by elements which are "undefinetely similar" each other. For this reason we call *clouds* the elements of the powerset of $\mathcal{P}(\mathcal{U})$.

Definition 2. Given a similarity $\mathcal{R}$ on $\mathcal{U}$, the *$\mathcal{R}$-singleton measure* is the fuzzy subset $\mu_{\mathcal{R}} : \mathcal{P}(\mathcal{U}) \rightarrow L$ defined by setting:

$$\mu_{\mathcal{R}}(X) = \bigwedge_{x,x' \in X} \mathcal{R}(x,x'). \text{ if } X \neq \emptyset$$
$$\mu_{\mathcal{R}}(\emptyset) = 1.$$

As an example, if $\mathcal{R}$ is a crisp equivalence relation then $\mu_{\mathcal{R}}(X) = 1$ if all the elements in X are pairwise equivalent, $\mu_{\mathcal{R}}(X) = 0$ otherwise. Is $\mathcal{R}$ is the identity relation, then $\mu_{\mathcal{R}}(X) = 1$ if X is a singleton, 0 otherwise. We have the following property:

Lemma 3. *Assume that $X \neq \emptyset$ and that $c \in X$. Then*

$$\mu_{\mathcal{R}}(X) = \bigwedge_{x \in X} \mathcal{R}(x,c).$$

Let $\mathcal{L}$ be a first order language whose set of constants we denote by $\mathcal{C}$ and whose set of predicate names we denote by $\mathcal{R}$ and denote by $\alpha : \mathcal{R} \to N$ the arity map. Suppose that both F_o and $\mathcal{C}$ are finite. Let X_d be a finite set of variables.

We call *set of extended terms* the set $\mathcal{T}_{X_d,\mathcal{C}} = X_d \cup \mathcal{P}(\mathcal{C})$ where $\mathcal{P}(\mathcal{C})$ is the powerset of $\mathcal{C}$. We call *set of extended closed terms* , the set $\mathcal{T}_\mathcal{C} = \mathcal{P}(\mathcal{C})$.

Given the terms $t_1, \ldots, t_n \in \mathcal{T}_{X,\mathcal{C}}$ and a constant $p \in \mathcal{R}$ such that $\alpha(p) = n$, we call *extended atomic formula* the expression $p(t_1, \ldots, t_n)$. If $t_1, \ldots, t_n$ are closed terms, then the expression $p(t_1, \ldots, t_n)$ will be called *extended fact* or *extended ground fact.* The extended terms and the extended patomic formulas will be called *extended first-order expressions.*

We assume as primitives two relations of similarity $eq_\mathcal{R}$ and $eq_\mathcal{C}$ on $\mathcal{C}$ and $\mathcal{R}$, respectively, where $eq_o(f, g) = 0$ if $\alpha(f) \neq \alpha(g)$ for any $f, g \in \mathcal{R}$.

We use eq_d, eq_o, $\mu_{eq_\mathcal{C}}, \mu_{eq_\mathcal{R}}$ to define a similarity relation on the set of extended terms $\mathcal{T}_{X,\mathcal{C}}$. We define a fuzzy binary relation eq as follows:

$$\begin{array}{rcl} eq(x,x) & = & 1 \\ eq(x,t) = eq(t,x) = 0 & & \text{if } x \neq t \text{ and } x \in X \\ eq(t,x) = eq(t,x) = 0 & & 0 \text{ if } x \neq t \text{ and } x \in X \\ eq(d,d') & = & \mu_\mathcal{C}(d \cup d') \text{ if } d, d' \in \mathcal{P}(\mathcal{C}). \end{array}$$

Where $\mu_\mathcal{C}$ denote $\mu_{eq_\mathcal{C}}$. For reasons of simplicity, we identify the elements x in $\mathcal{C}$ with the singleton sets $\{x\}$ in $\mathcal{P}(\mathcal{C})$. Therefore, we write $eq(x, x')$ instead of $eq(\{x\}, \{x'\})$ Furtherly, we extend the relation eq to the set of extended first-order formulas by setting

$$eq(p(t_1, \ldots, t_n), q(t'_1, \ldots, t'_n)) = \inf_{i=1..n} eq_\mathcal{R}(t_i, t'_i) \wedge eq_o(p, q).$$

The fuzzy relation eq is *not* a similarity relation on the set of first-order expressions. In fact, for any $d \in \mathcal{P}(\mathcal{C})$, $eq(d, d) = \mu(d) \neq 1$ in general. Noneontheless, the following proposition shows that the relation eq is transitive:

Proposition 4. *The binary relation eq is symmetric and transitive.*

2.2 Unification through Fuzzy Similarity Relations

We define a "soft" version of first-order unification. Classically, two first-order expressions t and t' are unifiable if there exists a substitution θ such that $\theta(t) = \theta(t')$. We call *system of equations* S a finite set of equations over extended first-order expressions. Given a similarity relation over first- order expressions, we can define a "degree of unifiability" of a set of term equations by interpreting in a multivalued logic the connectives of the meta-language used in the classical definition of unifiability:

a finite system of equations S is unifiable if there exists a substitution θ such that $\theta(t) = \theta(t')$ for any equation $t = t' \in S$.

Since the soft interpretation of the existential quantifier and of the universal quantifier is the supremum and the infimum respectively, the *degree of unifiability* of a set of equations S is defined by

$$U(S) = \sup_{\theta \in \Theta}\{ \inf_{t=t' \in S} eq(\theta(t), \theta(t'))\} \text{ for any } t, t' \in \mathcal{T}_{X_d,C}\}. \qquad (1)$$

Where Θ is the set of extended substitutions $\theta : X \rightarrow \mathcal{P}(C) \cup X$ Besides, for any substitution θ, we can define an *unification index of θ with respect to S* by putting

$$\nu(\theta, S) = \inf_{t=t' \in S} eq(\theta(t), \theta(t')).$$

Remark that from the degree of unifiability of a system of equations S we can easily define a binary fuzzy relation R_u expressing the unifiability degree of a pair of firs-order expressions: simply set $R_u(t,t') = U(\{t = t'\})$ if $t, t' \in \mathcal{T}_{X_d,C}$, otherwise, if $t = p(t_1, \ldots, t_n), t' = q(t_1, \ldots, t_n)\})$ then $R_u(t,t') = U(\{t_i = t'_i, i = 1 \ldots n\}) \wedge eq_o(p,q)$ where h is a function that yields the head of a predicate.

Many features of the classical unification relation still hold in the fuzzy counterpart; as an example, consider the aspects of unification connnected to computability theory. Assume that L and $\mathcal{T}_{X,C}$ are a computable lattice and a computable set, respectively. Then we can apply the concepts of L-computability theory (see [13]) and state whether or not the fuzzy relation U, seen as a fuzzy subset of the set of finite equations over $\mathcal{T}_{X,C}$, is decidable or less. We have the following result:

Theorem 5. *Let $Eq(S)$ be the set of equations over $\mathcal{T}_{X,C}$ Consider the fuzzy unification degree U as a fuzzy subset of $\mathcal{P}(Eq(S))$. Then, if eq_d is an L-computable subset of $C \times C$, U is L-computable.*

3 A Fuzzy Unification Algorithm

In this section we give an extension of the Martelli-Montanari algorithm for first-order terms unification ([15]). For a detailed description of this algorithm, see appendix A. We describe an algorithm taking a set of equations as input and giving a set of clouds and an extended unifier θ as output, i.e. a unifier whose range is the set of extended terms $\mathcal{T}_{X,C}$. The degree of unifiability of the unifier θ, written as $\nu(\theta)$, is assumed as the infimum among the eq_d and eq_o-singleton measures of the clouds.

In our algorithm the clouds are originated by an equation of the kind $a = b$ where a and b are constants in C; mismatches of this kind are caused either by the equations in input, as the one above, or by the links produced by the variables. To clarify this latter case, consider the equations $x = a, x = y, y = b$. Clearly, the variables x, y are linked in such a way to cause a "clash" among the constants a and b. While classical unification does not provide solutions for this case, in our approach we set up an extended unifier of the form $\theta = \{x = \{a, b\}\}$. The cloud $\{a, b\}$ will be increased each time a cloud containing a or b will be formed, or a link among variables instantiated with a or b will be produced. At the end of the algorithm, each cloud will be made up of constants that, if equal, would enable the algorithm to give a classical solution. For this reason, the unification index $\nu(\theta)$ can be interpreted as the degree with which the classical unification algorithm would have found a solution. In other words, $\nu(theta)$ is the "cost" one must pay to solve the classical unification problem for the input system of term equations.

The following example illustrates the main differences between the classical Martelli-Montanari unification algorithm and our extension.

Example 1. Let $S = \{p(a,b,c), p(x,x,x)\}$. In this case the Martelli Montanari unification procedure would yield the system of multiequations $Z = \{x = a = b = c\}$. Clearly, there is a constant mismatch, causing the algorithm to fail. In our approach, the multiequation $a = b = c$ is dealt as a cloud; therefore, a solution to the unification problem still exists, i.e. the extended unifier $\theta = \{\{x\} = \{a,b,c\}\}$, whose degree of unifiability $\nu(\theta)$ is $\mu_{eq_d}(\{a,b,c\})$, that could be assumed as the cost one must pay to collapse the cloud $\{a,b,c\}$ into a singleton set, say $\{a\}$ and allowing a solution to the classical unification problem for S.

The generalized unification algorithm is applied to a system of equations S. Such equations are converted in a "normalized" set S' such that $S' = \{X_1 = D_1, \ldots X_t = D_t\}$ where X_i is a (posibly empty) set of variables and D_i a cloud, respectively. The set S' is called *normalized* provided that the left-hand-side members and the right-hand-side members of its equations are respectively disjoint. A normalized set of multiequations has a straightforward implicit generalized unifier: for any equation of the kind $X = D$, simply take $\{x \rightarrow D\}$ for any variable $x \in X$. If $D = \emptyset$, the extended unifier is obtained by linking the variables in X one another. The extended unifier θ is associated to a "degree of unifiablility", given by $\nu(S) = \bigwedge_{i=1}^{t} \mu(D_i)$. In accordance with the kind of the clouds, μ can be μ_{eq_d} or μ_{eq_o}. In the following subsection we will prove some important properties of such a degree. Furtherly, we extend the generalized unifier by trying to expand the clouds without altering the cost payed for unification, i.e. the degree $\nu(S)$. This is a thoroughly algebraic operation, and involves only the fuzzy binary relation eq. We get the system S'' which is of the kind $\{X_1 = D'_1, \ldots X_t = D'_t\}$ where $D'_i \supseteq D_i$ for any i and $\nu(S'') = \nu(S')$. Finally, we normalize S'' and take $\nu(S'')$ as the degree of unificability of its implicit generalized unifier. Such an operation is called *algebraic lifting*.

To clarify this concept, the following example performs the algebraic lifting of the extended unifier of example 1.

Example 2. Let $C = \{a,b,c,d,e\}$, where eq_d is given as follows: $eq_d(a,b) = eq_d(a,d) = eq_d(b,d) = eq_d(b,c) = eq_d(c,d) = 0.9$
$eq_d(a,c) = 0.6, eq_d(e,x) = 0, \forall x \in C$. Consider the set of term equations $S = \{p(a,b,c), p(x,x,x)\}$. Its normalized form is $S'' = \{\{x\} = \{a,b,c\}\}$ and $\mu(\{a,b,c\} = 0.6$. Since $eq_d(d,a) = 0.9 \geq 0.6$, the cloud $\{a,b,c,d\}$ has the same eq_d-singleton measure μ of $\{a,b,c\}$. This means that the extended unifier $\theta' = \{x \rightarrow \{a,b,c,d\}$ solves the unification problem for S at the same cost of the unifier θ. Moreover,

the unifier θ' is a more significative unifier for S than θ, since it gives a huge clouds ad the same eq_d-singleton measure.

In the following we give a procedural description of the extended Martelli-Montanari algorithm of unification.

```
procedure  mgu($S, \theta, D$)
use        mgu1($Z, \theta, D$);
           lifting($S, T, D, Dz$);
begin
    $T := \emptyset$; $D := \emptyset$;$\theta = \epsilon$;
    $D' = \emptyset$;$S' = \emptyset$;
    $split(S, S', D, D')$;
    $Z := compact(convert(S'))$;
    $D := D \cup D'$;
    mgu1($Z, \theta, D$);
end;
```

```
procedure mgu($Z, \theta, D$);
begin
           for all    $S = \emptyset \in Z$      do $T := T \cup \{S = \emptyset\}$; $Z := Z \backslash \{S = \emptyset\}$;
           for all    $\emptyset = (a_1, \ldots, a_n) \in Z$  do $Z := Z \backslash \{\emptyset = (a_1, \ldots, a_n)\}$;
                      $D := D \cup \{(a_1, \ldots, a_n)\}$;
             for all    $S = M \in Z$       do begin(*plug-in*)
               $T := T \cup \{S = M\}$;
               $Z := Z \backslash \{S = M\}$;
                                       end(*plug-in_end*)
lifting($S$,T,D,Dz)
D := Dz;                    $\theta$ := solve(T)
end;
```

Algorithm 1

```
procedure          lifting(S,S',D,D')
       begin
       $S' := \emptyset$;
       $\lambda := \bigwedge_{d \in D} \mu(d)$
forall        $X = D \in S$  do
              $S' := S' \cup \{X = H_\lambda(D)\}$;
       $S'$ := convert($S'$);
       $D' := \cup_{X=D \in S'} \{D\}$;
       end
```

Algorithm 2

Split(S, S', D, D') is a procedure that takes a pair of systems of term equations S and S' over the set of first-order expressions, and a pair of sets of clouds D and D' as arguments and, when it terminates, S' is a system of equations obtained by decomposing along its arguments any equations of the kind $p = q$ where p and q are predicates with the same arity; D' is the normal form of the set of clouds produced by splitting any equation of the form $t = t'$ along its arguments; whenever an equation of the kind $t = t'$ is encountered, such that $\alpha(t) \neq \alpha(t')$, the algorithm *Split* yields a failure state $failure := true$ that is inherited by the whole unification algorithm.

Convert(S) is a function whose output is a system of multiequations. A multiequation is an expression of the form $x_1 = x_2 = \ldots = x_n = a_1 = a_2 = \ldots = a_m$ which can be conveniently represented in the form $\{x_1, x_2, \ldots, x_n\} = (a_1, a_2, \ldots, a_m)$.

The function convert simply maps each term equation $t = t'$ into an equivalent multiequation of the kind $X = D$, where X is a set of variables and D is a cloud. Either X or D could be empty. The procedure is specified as follows:

$$\text{Convert } (t, t') = \begin{cases} \{t, t'\} = \emptyset & \text{if } t, t' \in X_d \\ \{t\} = t' & \text{if } t \in X_d, t' \in \mathcal{P}(C) \\ \{t'\} = (t) & \text{if } t' \in X_d, t' \in \mathcal{P}(C) \\ \emptyset = t \cup t' & \text{if } t, t' \in \mathcal{P}(C) \\ \emptyset \text{ if } t, t' \in \mathcal{P}(C) \text{ and } t = t' & \end{cases}$$

Compact(Z) is a function that takes a system of multiequations as input and yields a "compactified" version of it. The compactification is performed through a merging of equivalence classes of multiequations generated by the following binary relation R

$$(S = M)R(S' = M') \text{ iff } (S \cap S' \neq \emptyset) \text{ or } (M \cap M' \neq \emptyset) \qquad (2)$$

Let $\bar{R}$ be the transitive closure of R. The relation $\bar{R}$ partitions a set of multiequations Z into a set of equivalence classes.

As a result, the function Compact gives a set of multiequations, each one made up of the merging, member by member, of the multiequations in the corresponding class.

Solve(S) is a function that takes a triangular set of multiequations as argument and derives an extended substitution as follows:

$$solve(\{x_1, \ldots, , x_n\} = M = \{x_1 \to M, \ldots, x_n \to M\}$$
$$solve(\{x_1, \ldots, x_n\} = \emptyset) = \{x_1 \to x_2, \ldots, x_{n-1} \to x_n\}$$

Where M is a cloud of constants in $\mathcal{P}(C)$.

The following example illustrates the behaviour of Algorithm 1:

Example 3. Let $S = \{p(a,b,c), p(x,x,x)\}$. In this case Split($S$) yields the set of equations $S' = \{x = a, x = b, x = c\}$. The function compact(Convert(S')) yields the set of multiequations $\{\{x\} = \{a,b,c\}\}$, where $\{a,b,c\}$ is a cloud. Then, the mgu($\{\{x\} = \{a,b,c\}\}, \theta, D$) is called, with $T = \emptyset$ and $D = \emptyset$. At the end of the main procedure mgu, the variable D is set with the cloud $\{a,b,c\}$ and θ is the generalized substitution $\{x \rightarrow \{a,b,c\}\}$. Considering the fuzzy similarity relation of example 2, we apply the operator $H_{0.6}$ to the cloud $\{a,b,c\}$, getting the cloud $\{a,b,c,d\}$. So the compactified version of S' is $\{\{x\} = \{a,b,c,d\}\}$. Finally, the cloud produced by the algorithm is $\{a,b,c,d\}$.

3.1 Algebraic Extension of Unifiers

The unifier supplied by the normalization of the input system of equation S goes through the process of algebraic lifting. This process is described as follows. Given a normalized system S, suppose $S = \{X_1 = D_1, \ldots, X_t = D_t\}$ and $\mu(D_i) = \lambda_i, i = 1, \ldots, t$. Consider the infimum $\lambda = \bigwedge i = 1^t \lambda_i$. We now partition the set of indexes $1 \ldots t$ into a family of indexes $I_1, \ldots, I_s$ such that, for all $j = 1 \ldots s$ $H_\lambda(D_m) \cap H_\lambda(D_n) \neq \emptyset$ for any $n, m \in I_j$. Consider the set $K_1 = \bigcup_{j_1 \in I_1} H_\lambda(D_{j_1}), \ldots, K_s = \bigcup_{j_s \in I_s} H_\lambda(D_{j_s})$. We have the following lemmas:

Lemma 6. *Let $H_\lambda(X)$ be an operator from $\mathcal{P}(\mathcal{C})$ into itself, defined as follows:*

$$H_\lambda(X) = \{y \in \mathcal{C} | eq(y,x) \geq \lambda \textit{ for some } x \in X\}.$$

Then H_λ is a closure operator, i.e.

i) $H_\lambda(X) \supseteq X$ for any X in $\mathcal{P}(\mathcal{C})$.
ii) $H_\lambda(Y) \supseteq H_\lambda(X)$ if $Y \supseteq X$.
iii) $H_\lambda(H_\lambda(X)) = H_\lambda(X)$

Lemma 7. *Let D be a cloud in $\mathcal{P}(\mathcal{C})$. Suppose that $\mu(D) = \lambda$. Then the following properties hold:*

(i) $\mu(H_\lambda(D)) = \lambda$
(ii) $H_\lambda(H_\lambda(D)) = H_\lambda(D)$.

Lemma 8. *For any $i = 1 \ldots t$, $\mu(H_\lambda(D_i)) \geq \lambda$*

Given the element λ, it is reasonable to "close" the clouds of bindings $D_1, \ldots, D_t$ by means of the operator H_λ, since λ is the "minimum cost" one must pay for having a solution to the unification problem. Such process is called *solution lifting*, since the cloud is maximized according to the criterion of the "similarity cost". The system $\bar{S} = \{X_1 = K_1, \ldots, X_s = K_s\}$ is the normalized version of the system S'. Remark that the normalization involves only clouds, since the sets of variables are already disjoint in S'. At this point, the lifting process proceeds along an alternate use of application of operator H_λ and a compactification operation, according to an algorithm of the following kind:

```
1. $S_0 = S'$;
2. $\lambda_0 = \lambda = \bigwedge_{i=1}^{t} \mu(D_i)$;
3. Until $\lambda_i = \lambda_{i+1}$ do begin
4. $S_{i+1} = compact(H_{\lambda_i}(S_i))$
5. $\lambda_{i+1} = \bigwedge_{D \in S_i} \mu(D)$
6. end until
```

Algorithm 3

The following theorem shows that Algorithm 3 is a process that is performed in an uniform number of steps

Theorem 9. *Let $S = \{X_1 = D_1, \ldots, X_t = D_t\}$ be a normalized system of term equations, such that $\bigwedge_{i=1}^{t} \mu(D_i) = \lambda$. Let $K_1, \ldots, K_s, s \leq t$, be defined as above. Let $S_1 = \{X'_1 = K_1, \ldots, X'_s = K_s\}$ be the lifted and compactified version of S. Then the following properties hold*

i) $H_\lambda(K_i) = K_i$ *for any* $i = 1, \ldots, s$
ii) $\bigwedge_{l=1}^{s} \mu(K_l) = \lambda$
iii) $S_i = S_1$, $\lambda_i = \lambda$ *for any* $i \leq 1$.

Theorem 9 shows that Algorithm 3 is actually a two-steps procedure. In other word, the system $S_1 = compact(H_\lambda(S')) = S'$ and $\lambda_1 = \lambda_0 = \lambda$. Therefore, the lifting process is simplifies into a one-step application of the operator H_λ and the compactification of the resulting system of equations, as implemented into algorithm 1.

4 Conclusions and Future Developments

The modelization of similarity-based reasoning is a useful tool, both from the theoretical point of view, since it enhances the scope of the classical logical systems, and for practical purposes, since AI-based applications deeply benefit of the features provided by tools that manage similarity. Our extension of first-order unification using a fuzzy relation of similarity supplies an elegant and powerful unification procedure that can be suited for resolution-based inferential engines to be used in deductive-data bases. We are aiming at extending this algorithm to the full-fledged algebra of first-order terms.

References

1. F.Arcelli, F.Formato and G.Gerla, Similarity-based Unification as a Foundation as a Fuzzy Logic programming, *Proceedings of the First Workshop on Logic Programming and Soft Computing*, Bonn, Germany, sept. 1996.
2. J.F.Baldwin, T.P.Martin and B.W. Pilsworth, The implementation of FPROLOG - A Fuzzy Prolog Interpreter, *Fuzzy Sets and Systems*, 23:119-129, 1987.
3. J.F.Baldwin, T.P.Martin and B.W. Pilsworth, *FRIL - Fuzzy and Evidential Reasoning in AI*, Research Studies Press, 1995.

4. L.Biacino and G.Gerla , Recursively enumerable L-subsets, in*Zeitschrift f. Logik und Grundlagen d. Math.*, (1989), 49-62.
5. M.K.Chakraborty, Graded Consequence: Further Studies, *Journal Studies of Applied non-Classical Logics*, 21, 1995.
6. D.Dubois, J.Lang and H.Prade, Towards Possibilistic Logic Programming, *Proceedings of ICLP 91*, Paris, 1991.
7. G.Escalada-Imaz and F.Manya, A linear interpreter for logic programming in ultivalued propositional logic, in *Proceedings of IPMU'94*, pp.151–156, Paris, 1994.
8. H.Kikuchi and M.Mukaidono, PROFIL - Fuzzy Interval Logic Prolog, *Preprints of Internation workshop on Fuzzy Systems Appplications*, pp.:205-206, 1988.
9. M.Ishizuka and N.Kanai, Prolog -ELF Incorporating Fuzzy Logic, in *Proceedings of 9th IJCAI*, 1985, pp.::701-703.
10. R.C.T.Lee, Fuzzy Logic and the Resolution Principle, *Journal of the ACM*, 19:109-119, 1972.
11. A. Martelli and U.Montanari, An Efficient Unification Algorithm, *ACM Transactions on Programming Languages and Systems*, Vol 4, No 2, April 1982.
12. M.Mukaidono, Z.Shen and L.Ding, Fundamentals of Fuzzy Prolog, *Int.Jornal of Approximated Reasoning*, 3,2, 1989.
13. L:Sterling and E.Shapiro, *The Art of Prolog*, MIT PRESS, 1987.
14. J.A.Robinson, A Machine-oriented logic based on resolution principle, in *JACM*, 12(1), pp.23–41, Jan. 1983.
15. H.Virtanen, A Study in Fuzzy Logic Programming, in *Proceddings of the 12th European meeting on Cybernetics and Systems'94*, pp. 249–256, Austria, April 1994.
16. H.Y.Yasui, Y.Hamada, M.Mukaidono, Fuzzy prolog based on Lukasiewicz implication and bounded product, in *Proc.IEEE Intern.Conf. on Fuzzy Sets*,2:949-954, 1995.
17. M.S.Ying, A Logic for Approximated Reasoning, *The Journal of Symbolic Logic*, 59, 1994.
18. L.Zadeh, Fuzzy Sets, *Information and Control*, 8, 1965.

This article was processed using the LaTeX macro package with LLNCS style