Similarity-based Logic Programming: a Fuzzy Resolution Rule

F.Arcelli Fontana
DIIIE, University of Salerno, 84084 Fisciano (SA), Italy.
email: arcelli@ponza.dia.unisa.it

F. Formato and G.Gerla
DIIMA University of Salerno, 84084 Fisciano (SA), Italy.
email: {formato, gerla}@ponza.dia.unisa.it

Abstract

We propose an extension of logic programming paradigm based on similarity. Starting from a new fuzzy unification algorithm based on similarity developed in [2], here we introduce a fuzzy resolution rule based on this extended unification. Essentially, our proposal of extended resolution is a replacement of the classical most general unifier (mgu) in the unification step for its extended counterpart. In our approach, unification fades into a "unification degree" on account of the introduction of a similarity in a first-order language with no constants. Intuitively, the unification degree of a set of first-order terms is the "cost" one must pay to consider the terms as "syntactically equal". For this reason, our extension of resolution yields a more structured kind of computed answer substitution, i.e. when the empty clause is reached a set of constants is also determined. The refutation degree of a goal formula is the degree by which such a set sharpens into a set of singletons.

1 Introduction

In the field of databases and information retrieval there is an ever increasing demand for systems able to deal with flexible queries and answers. Deduc-

tive databases, in particular, must cope with approximate inference and more flexible ways of information retrieval. In this work we extend classical resolution, and consequently, classical refutation, into a more structured process which is particularly significant when, in the classical counterpart, a finite failure occurs. Such a technique could be usefully exploited for query evaluation in deductive databases. This is the main reason why we consider languages where no function symbol occurs. Our extension of resolution is basically a replacement of the classical unification step for an extended version of unification, where the search for an unifier of a set of first-order terms turns into the computation of a "unification degree". The unification degree of a set of terms is the "cost" one must pay to consider a set of constants, that we call "clouds", as sharpened into a singleton; it is obtained introducing a similarity into a first-order language. The set of constants, called "set of constraints", is a collection of constants that, in the process of classical unification, cause the unification to fail. Our context is an extension of Horn logic for definite logic programs. We formulate the extension of the classical theory defined in [1] and we derive the extension of resolution accordingly, collecting the set of conditions stemming from each step of unification and eventually determining its "co-diameter", i.e. the degree to which this set sharpens into a set of singletons. Such a number is the degree of derivation of the goal formula.

The paper is organized through the following sections. In section 2 we introduce similarity in a first-order language, we define the concept of cloud, a particular operator on clouds and the principal features of an extended similarity-based language. In section 3 we define an extended resolution rule through a set of constraints. Finally in section 4 we give a simple example of a fuzzy logic programming system based on the proposed extension of resolution rule.

1.1 Related Works

Different fuzzy extensions of the resolution rule of [12] used in classical logic programming have been proposed in the fuzzy context. The first work on fuzzy resolution for a set of ground clauses has been given by Lee [9]; several interesting properties of resolution in fuzzy logic can be found in subsequent works as for example in [4], resolution has been generalized to general clauses involving universally quantified variables and the problem of search the "more informative proof" has been pointed out. In [11] and [13] a fuzzy

resolution principle is defined at first for propositional logic and then for full first-order logic, with the proof of its completeness and the description of a fuzzy prolog system based on it. An extension of the resolution principle to possibilistic logic, where clauses are weighted with a degree which represents a lower-bound of a necessity or possibility measure is described in [7] and resolution in the context of possibilistic logic with fuzzy constants is discussed in [8]. While in [6] and [5] fuzzy resolution has been defined in the context of evidential logic. In [15] a λ-resolution method is proposed and its completeness proved in the context of operator fuzzy logic in which the fuzzy degree of propositions is directly represented. Virtanen in [14] shows how SLD-resolution used in logic programming can be modified and used for Linguistic Logic Programming, where variables may take fuzzy terms. This kind of resolution is based on the notion of fuzzy equality. In [16] a refutation method based on tree resolution is proposed where truth value are assigned to the program clauses based on adapted Lukasiewicz implication.

In most cases the fuzzy extension of the resolution rule, doesn't imply to use a fuzzy extension of the unification algorithm and classical unification is used (based on truth functionality). In other cases it is necessary to use fuzzy unification methods carried out in several ways.

2 Introducing Similarity in a First Order Language

The theoretical foundations of the fuzzy resolution rule which we propose are based on the notion of similarity. Similarity is a many-valued extension of the classical notion of equality.

Definition 1 *A* similarity *on a domain $\mathcal{U}$ is a fuzzy subset $\mathcal{R} : \mathcal{U} \times \mathcal{U} \rightarrow [0, 1]$ of $\mathcal{U} \times \mathcal{U}$ such that the following properties hold:*

i) $\mathcal{R}(x, x) = 1$ for any $x \in \mathcal{U}$

ii) $\mathcal{R}(x, y) \geq \mathcal{R}(y, x)$ for any $x, y \in \mathcal{U}$

iii) $\mathcal{R}(x, z) \geq \mathcal{R}(x, y) \wedge \mathcal{R}(y, z)$ for any $x, y, z \in \mathcal{U}$.

The operators $\wedge$ and $\vee$ correspond to the least upper bound (lub) and the greatest lower bound (glb) operators respectively. Given a similarity $\mathcal{R}$,

we can associate to every set X of elements of $\mathcal{U}$ a number $\mu(X)$ expressing the extent to which all the elements contained in X are similar.

$$\mu(X) = \begin{cases} \bigwedge\limits_{x,x' \in X} \mathcal{R}(x,x') \text{ if } X \neq \emptyset \\ 1 \qquad\qquad\qquad \text{otherwise.} \end{cases}$$

The number $\mu(X)$ is called *co-diameter of* X. We call "clouds" the finite subsets of $\mathcal{U}$ and therefore we say that $\mu(X)$ is the co-diameter of the cloud X. A *system of clouds* is a finite set of clouds. Given a system of clouds $Z = \{X_1, ...X_n\}$ we call *co-diameter of* Z the number

$$\xi(Z) = \bigwedge_{i=1}^{n} \mu(X_i).$$

We use the term "co-diameter" since the function $\mathcal{R}'(x,y) = 1 - \mathcal{R}(x,y)$ is a distance whose related diameter is the function $\delta(X) = 1\text{-}\mu(X) = \bigvee\limits_{x,y \in X} \mathcal{R}'(x,y)$. The following properties of μ are given.

Proposition 1 *Let* $\{X_i\}_{i=1..n}$ *be a finite sequence of clouds such that, for any* $i = 1..n-1$, $X_i \cap X_{i+1} \neq \emptyset$.

$$\mu\left(\bigcup_{i=1}^{n} X_i\right) = \bigwedge_{i=1}^{n} \mu\left(X_i\right).$$

Moreover, assume that $X \neq \emptyset$ *and that* $c \in X$. *Then*

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

Proof. Clearly, $\bigwedge\limits_{x \in X} \mathcal{R}(x,c) \geq \mu(X)$. Moreover, for any $x, y \in X$

$$\mathcal{R}(x,y) \geq \mathcal{R}(x,c) \wedge \mathcal{R}(c,y) \geq \bigwedge_{x \in X} \mathcal{R}(x,c).$$

Therefore

$$\mu(X) \geq \bigwedge_{x \in X} \mathcal{R}(x,c).$$

Q.E.D.

2.1 An Operator Over Systems of Clouds

We say that a system of clouds Z is *compact* if its elements are pairwise disjoint. The following operator will be used to obtain a compact system of clouds.

Definition 2 *Let $Z = \{M_1, ..., M_n\}$ be a system of clouds. The following is an operator on the clouds of Z*

$$T_Z(M) = \bigcup_{M' \cap M \neq \emptyset, M' \in Z} M'.$$

The extension of T_Z to a system of clouds is as follows:

$$T_Z(Z) = \{T_Z(M) \mid M \in Z\}$$

The operator T_Z does not change the co-diameter of a system of clouds. The operator T_Z can be iterated in the usual manner, by setting $T_Z^0(Z) = Z$, $T_Z^1(Z) = T_Z(Z)$ and $T_Z^{n+1}(Z) = T_Z(T_Z^n(Z))$ for any integer $n \geq 1$.

Definition 3 *Let Z be a system of clouds. We define* degree *of the system Z the number $\varphi(Z)$ defined as follows:*

$$\varphi(Z) = \min_n \{T_Z^n(Z) = T_Z^{n+1}(Z)\}.$$

Since Z is a system of finite clouds, the degree of Z is a finite non-negative integer. Obviously, if Z is a compact system of clouds, $\varphi(Z) = 0$ and Z is a fix point of T_Z,where for a compact system we mean a system where no cloud overlaps.

Definition 4 *Let Z be a system of clouds. The operator Compact is defined as follows:*

$$Compact(Z) = T_Z^{\varphi(Z)}(Z).$$

By definition of $\varphi(Z)$ the operator *Compact* transforms a system of clouds into a compact system. We have the following properties which show that the *Compact* operator keeps the co-diameter invariant.

Proposition 2 *Let Z be a system of clouds. Then*

i) $\xi(Compact(Z)) = \xi(Z)$

ii) $Compact(Compact(Z)) = Compact(Z)$.

Proof. *i*) Immediate by definition of co-diameter and by Proposition **??**.
ii) Immediate, since $Compact(Z)$ is compact.

Q.E.D.

2.2 An Extended Similarity-Based Language

We use a fuzzy similarity relation to extend both first-order unification and resolution. To do that, let $\mathcal{L}$ be a classical first-order language and denote

-by $\mathcal{V}$ the *set of variables*

-by $\mathcal{C}$ the *set of constants*

-by P the *set of predicate symbols*

-by $\mathcal{T}_{\mathcal{V},\mathcal{C}}$ the *set of terms*, i.e. the set $\mathcal{V} \cup \mathcal{C}$.

We expand the language $\mathcal{L}$ into the language $\mathcal{L}'$ whose set of constants is the set $\mathcal{C}' = \mathcal{P}(\mathcal{C}) - \{\emptyset\}$which is called *set of extended constants*.

In the sequel, we use extended-*type* (denoted by *e-type*), where *type* is in turn a *constant,* a *term* and a *formula* of $\mathcal{L}'$. Also, we denote with $\mathcal{T}_{\mathcal{V},\mathcal{C}'}$ the *set of e-terms*, i.e. the set $\mathcal{V} \cup \mathcal{C}'$. An *extended substitution* (briefly, an *e-substitution*) is a map $\theta : \mathcal{V} \to \mathcal{T}_{\mathcal{V},\mathcal{C}'}$. We denote with Θ the set of *e*-substitutions. Finally, an *atomic extended formula*, or *atomic e-formula,* will be an expression of the kind $p(t_1, ..., t_n)$, where $p \in \mathcal{P}$ and $t_i \in \mathcal{T}_{\mathcal{V},\mathcal{C}'}$ for any $i = 1, ..., n$.

We identify any constant c in $\mathcal{C}$ with the singleton $\{c\}$ in $\mathcal{C}'$. Given two similarities over P and $\mathcal{C}'$,denoted respectively as $\mathcal{R}_P$ and $\mathcal{R}_\mathcal{C}$ we define a relation $\mathcal{R}$ on $P \cup \mathcal{C}' \cup \mathcal{V}$ in the following way:

$$\mathcal{R} = \mathcal{R}_P \cup \mathcal{R}_\mathcal{C} \cup \mathcal{R}_\mathcal{V}$$

where $\mathcal{R}_\mathcal{V}$ is a similarity on $\mathcal{V}$ defined as follows:

$$\mathcal{R}_\mathcal{V}(x, y) = 0 \ \ if \ x \neq y \text{ and } x, y \in \mathcal{V}$$

$$\mathcal{R}_\mathcal{V}(x, x) = 1.$$

We define eq^* on the set of atomic e-formulas by setting:

$$eq^*(p(t_1,\ldots,t_n),q(t'_1,\ldots,t'_m)) = 0 \ \ if \ \ n \neq m$$

$$eq^*(p(t_1,\ldots,t_n),q(t'_1,\ldots,t'_n)) = \mathcal{R}(p,q) \wedge (\bigwedge_{i=1..n} \mathcal{R}(t_i,t'_i)) \text{ otherwise.}$$

This fuzzy relation is symmetric and transitive, as it is easy to prove, but it is *not* a similarity since it is not reflexive, in general.

To any compact system of clouds Z in $\mathcal{T}_{V,\mathcal{C}'}$ we associate an e-substitution θ_Z defined as follows:

$\theta_Z(x) = x$ if $x \notin M_1 \cup ... \cup M_n$;

$\theta_Z(x) = M_i \cap \mathcal{C}$ provided that $x \in M_i$ and $M_i \cap \mathcal{C} \neq \emptyset$;

$\theta_Z(x) = x_j$ provided that $x \in M_i$, $M_i \cap \mathcal{C} = \emptyset$ and x_j is a variable in M_i.

As an example, we can assume that x_j is the first variable in the sequence $x_1, x_2, ...$ belonging to M_i. Given a system of cloud $Z = \{M_1, ..., M_n\}$, we write a generic cloud M in Z in the form $X \sqcup D$,where $X = M \cap \mathcal{V}$ and $D = M \cap \mathcal{C}$. Analogously, given an e-substitution θ, by $\theta(M)$ we denote the cloud $\bigcup_{x \in X} \theta(X) \sqcup D$, identifying a variable y with its singleton $\{y\}$where necessary.

Finally we define the operator $Ground$ in the following way: given a cloud $X \sqcup D$, we set $Ground(X \sqcup D) = D$; given a set of clouds Z, we set $Ground(Z) = \{Ground(M) \mid M \in Z\}$.

Finally $t\dot{\sqcup}t'$ is a cloud defined as follows:

$$t\dot{\sqcup}t' = \begin{cases} t \cup t' \text{ if } t \text{ and } t' \in \mathcal{C}' \\ \{t\} \cup t' \ \text{ if } t \in \mathcal{V}, t' \ \in \mathcal{C}' \\ \{t'\} \cup t \text{ if } t \in \mathcal{C}', t' \ \in \mathcal{V} \\ \{t, t'\} \text{ if } t, t' \in \mathcal{V}. \end{cases}$$

We consider two functions $Trans\_t$ and $Trans\_p$ transforming a system of e-equations into systems of clouds in $\mathcal{C} \cup \mathcal{V}$ and $\mathcal{P}$, respectively. We give the following functions:

$$\begin{aligned} Split\_t(p(t_{1,},...,t_m) &= q(t'_1,...,t'_m)) = \{t_1\dot{\sqcup}t'_1,...,t_m\dot{\sqcup}t'_m\} \\ Split\_p(p(t_{1,},...,t_m) &= q(t'_1,...,t'_m)) = \{p,q\}. \end{aligned}$$

Moreover, if $S = \bigcup_{i=1}^{n} \{e_i = e'_i\}$ we set

$$
\begin{aligned}
Trans\_t(S) &= \bigcup_{i=1...n} Split\_t(e_i = e'_i) \\
Trans\_p(S) &= \bigcup_{i=1...n} Split\_p(e_i = e'_i).
\end{aligned}
$$

3 Extended Resolution as a set of Constraints

We start from a definition of extended resolution which is a straightforward extension of the classical resolution for definite programs given in [1]. A *definite (logic) e-program* P is defined according to its classical counterpart, i.e. it is a finite set of *e-clauses, namely* e-formulas of the kind $A \leftarrow B_1, ..., B_n$ where C and $B_{i,\ i=1,...,n}$ are atomic e-formulas. *A negative e-clause* is an e-formula of the kind $\neg A_1 \vee ... \vee \neg A_k$, where A_i are atomic e-formulas.

We first introduce the concept of "e-resolvent". As usual, a *variant* of an e-clause C in a definite e-program P will be a renaming of the free variables occurring in C.

Definition 5 *Let P be a definite e-program, let $N = \neg A_1 \vee ... \vee \neg A_k$ be a negative e-clause and let $C = A \leftarrow B_1, ..., B_n$ be a variant of an e-clause in P. The negative e-clause*

$$N' = \leftarrow \theta(A_1, ..., A_{i-1}, B_1, ..., B_n, A_{i+1}, ..., A_m)$$

is called an e-resolvent of N and C with e-mgu θ, and set of constraint $Cond$ provided that e-formula A_i unifies with the atomic e-formula in the head of C according to the similarity based unification algorithm described in **??**, *yielding an e-mgu θ and a set of constraints, $Cond = Ground(Compact(Trans\_t(S)) \cup Compact(Trans\_p(S))$ where $S = \{A = Head(C)\}$.*

We are now ready to give the following definition of extended SLD derivation.

Definition 6 *Let P a definite e-program and let N be a negative e-clause. An extended SLD-derivation of $P \cup \{N\}$, briefly e-SLD derivation is a maximal sequence of negative e-clauses N_i, a sequence of variants of an e-clause C_i, a sequence of e-substitutions θ_i, a sequence of systems of e-equations S_i and a sequence of systems of clouds $\mathcal{C}_i$ such that, for any i :*

-$N_0 = N$

-N_{i+1} *is an e-resolvent of* N_i *and* C_i *with* θ_i *as e-mgu and* $\mathcal{C}_i$ *is a set of constraints.*

- $S_i = \{A_i = B_i\}$ *where* A_i *is an atomic e-formula in* N_i *and* B_i *is the atomic e-formula in the head of the clause* C_i.

-$\mathcal{C}_i = Ground(Compact(Trans\_t(S_i)) \cup Compact(Trans\_p(S_i))$

-θ_i *is an e-mgu of the system of e-equations* S_i*; in particular,* $\theta_i = \theta_{S_i}$

-C_i *has no variable in common with* $N_0, C_0, ..., C_{i-1}$.

If, for some integer $\bar{n}$, $N_{\bar{n}}$ is empty, then the e-SLD derivation halts and it is called e-*SLD refutation.* In this case the set $\bigcup_{i=1,..,\bar{n}} \mathcal{C}_i$ is called the *constraint-set* of the e-SLD refutation. Still, the restriction of e-substitution $\theta_{\bar{n}} \circ \theta_{\bar{n}-1} \circ ...\theta_1$ to the variables in N will be called *e-computed answer* substitution for $\{N\} \cup P$.

If an e-SLD derivation is finite and it is not an e-refutation, it is called a *failed* derivation. The e-clauses C_i are called *input e-clauses.*

Given an e-refutation $r_{P,N}$ of $P \cup \{N\}$,halting after $\bar{n} + 1$ steps, we call *degree* of the e-refutation of $\{N\} \cup P$, and we denote it $h(r_{P,N})$, the number $\xi(Compact(\bigcup_{i=0,..,\bar{n}} \mathcal{C}_i))$. In the following we give an example of e-SLD refutation.

Example 1 *Let* P *be the following program:*

$$
\begin{array}{l}
p(x,y) \leftarrow q(x), r(y). \\
r(x) \leftarrow s(x,y), s(y,z). \\
q'(a). \\
s'(a,c). \\
s''(e,f).
\end{array}
$$

Let $N_0 = p(x,y)$

we have the following e-SLD derivation of $P \cup \{N\}$

$$
\begin{array}{rcl}
N_0 & = & p(x,y), \\
C_0 & = & p(x_1,y_1) \leftarrow q(x_1), r(y_1) \\
\theta_0 & = & \{x \rightarrow x_1, y \rightarrow y_1\} \\
S_0 & = & \{p(x,y) = p(x_1,y_1)\} \\
\mathcal{C}_0 & = & \{\{p\}\}
\end{array}
$$

$$
\begin{array}{rcl}
N_1 & = & q(x_1), r(y_1) \\
C_1 & = & q'(a). \\
\theta_1 & = & \{x_1 \rightarrow a\} \\
S_1 & = & \{q(x_1) = q'(a)\} \\
\mathcal{C}_1 & = & \emptyset
\end{array} \tag{1}
$$

$$
\begin{array}{rcl}
N_2 & = & r(y_1) \\
C_2 & = & r(x_2) \leftarrow s(x_2, y_2), s(y_2, z_2) \\
S_2 & = & \{r(y_1) = r(x_2)\} \\
\theta_2 & = & \{y_1 \rightarrow x_2\} \\
\mathcal{C}_2 & = & \emptyset
\end{array}
$$

$$
\begin{array}{rcl}
N_3 & = & s(x_2, y_2), s(y_2, z_2) \\
C_3 & = & s'(a, c). \\
\theta_3 & = & \{x_2 \rightarrow a, y_2 \rightarrow c\} \\
S_3 & = & \{s(x_2, y_2) = s'(a, c)\} \\
\mathcal{C}_3 & = & \{\{a\}, \{c\}, \{s, s'\}\}
\end{array}
$$

$$
\begin{array}{rcl}
N_4 & = & s(c_2, z_2) \\
C_4 & = & s''(e, f)\theta_4 = \{z_2 \rightarrow f\} \\
S_4 & = & \{s(c_2, z_2) = s''(e, f)\} \\
\mathcal{C}_4 & = & \{\{e, c\}, \{s, s''\}\} \\
R_4 & = & s(c, z_2)
\end{array}
$$

$$C_4 \quad = \quad s''(e,f)$$

In this case the e-SLD resolution chain is an e-refutation and the e-computed answer substitution of the e-refutation is $\theta = \{x \rightarrow a, y \rightarrow a\}$. *The degree of the e-refutation is*

$$\xi(Compact(\bigcup_{i=0,..,4} \mathcal{C}_i)) = \xi(\{\{e,c\},\{s,s',s''\},\{q,q'\},\{f\},\{c\},\{a\}\})$$

This is the cost turn to pay to flatten into a set of singletons the set of constraints and make a classical refutation out of an e-refutation.

A negative e-clause N may have several e-refutations with respect to a given definite e-program P.

Definition 7 *Let* $\mathcal{R}(P,N)$ *be the set of e-refutations of* $\{N\} \cup \{P\}$. *We say that a negative e-clause* N *is derivable from* P *with degree* λ *if*

$$\lambda = \bigvee_{r \in \mathcal{R}(P,N)} h(r).$$

4 Introducing similarity in a deductive data base

Deductive databases can be viewed as information retrieval systems where knowledge is stored both explicitly, through a record-based data structure, both implicitly, through a set of rules that specify data in an intensional manner. Data are extracted through a "query evaluation" method. The introduction of a similarity in a database gives a certain flexibility, both in terms of data and query.

The extended resolution described in the previous section is a flexible evaluation method for answering queries that cannot be satisfied in the classical case.

For example, consider the following classical database of books of different kinds with some associated rules:

```
adventurous(isle_of_treasure).
adventurous(murder_on_orient_express).
adventurous(james_bond).
```

`horror(drakula).`
`good(X) :- interesting(X),cheap(X).`
`cheap(X) :- cost(X,L), L =< 10.`

The goal `good(X)`has no classical solution, since no fact in the database unifies with the atom `interesting(X).` Nevertheless, it seems reasonable to consider the constants `interesting` and `adventurous` as "similar" to a certain degree.

More precisely, consider the following similarity relation $\mathcal{R}$, evaluated using any continuous t-norm for the following conjunctions:

-$\mathcal{R}(\texttt{adventurous}, \texttt{interesting}) = 0.9$

-$\mathcal{R}(\texttt{adventurous}, \texttt{horror}) = 0.7$

-$\mathcal{R}(\texttt{horror}, \texttt{interesting}) = 0.7$

-$\mathcal{R}(a, b) = 0$ for any $a, b \in \{$`murder_on_orient_express`,
`isle_of_treasure,james_bond,drakula`$\}$

The query `good(X)` can be evaluated with an e-SLD derivation,

giving as a result the following pairs of computed e-substitution answers and constraint sets: $<\{$`X`$\rightarrow$ `James_Bond`$\}, \{$`adventurous`, `interesting`$\} >$,

$<\{$`X`$\rightarrow$`isle_of_treasure`$\}$,
$\{$`adventurous`, `interesting`$\} >$,
$<\{$`X`$\rightarrow$ `murder_on_orient_express`$\}$,
$\{$`adventurous`, `interesting`$\} >$,
$< \{$`X`$\rightarrow$ `dracula`$\}, \{$`horror`, `interesting`$\} >$.

This means that the query `good(X)` yields , for example, the answer `isle_of_treasure` under the condition that the genre `horror` is similar to `interesting`; in other words, the co-diameter $\xi(\{$`horror`, `interesting`$\})$ of the cloud $\{$`horror`, `interesting`$\}$ represents the cost one must pay to have a classical refutation of `good(X)` with `X->isle_of_treasure` as computed answer substitution.

5 Future Developments

In this work we have briefly described a new kind of fuzzy resolution based on similarity. We introduced a similarity in a first-order language and then, starting from the extension of unification theory, we proposed a mechanism of extended resolution. The main purpose of this is building the kernel of a fuzzy logic programming system under development. We are currently going to study more informative forms of computed answer substitutions,

according to the set of conditions yield in the extended refutation and to study the operational and declarative semantics of such an extension of logic programming. Moreover, we are also planning to use other t-norms for the similarity, such as the natural product in $[0, 1]$ and the Lukasiewitz sum.

References

[1] K.R.Apt. Logic programming. In *Handbook of Theoretical C.S.,* Chapter 10, vol B, J.van Leeuvwen Ed., Elsevier Science, 1990, pp. 495-574.

[2] F.Arcelli, F.Formato and G.Gerla. Similarity-based Unification. *Submitted to FSS Int.Journal,* may 1997.

[3] F.Arcelli, F.Formato and G.Gerla. Fuzzy Unification as a Foundation of Fuzzy Logic Programming. In *Logic Programming and Soft Computing*, Research Studies Press, Uncertain Theory in Artificial Intelligence Series, 1998.

[4] A.R.Aronson, B.E.Kacobs and J.Minker. A note on fuzzy deduction. In *JACM*, Vol.27, No.4, 1980, pp.599-603.

[5] J.F.Baldwin, T.P.Martin and B.W. Pilsworth. *FRIL - Fuzzy and Evidential Reasoning in AI.* Research Studies Press, 1995.

[6] J.F.Baldwin. Evidential Support Logic Programming. *Fuzzy Sets and Systems*, 24:1-26, 1987.

[7] D.Dubois, J.Lang and H.Prade. Towards Possibilistic Logic programming. In *Proceedings of ICLP91*, Paris, 1991.

[8] D.Dubois, J.Lang and H.Prade. Automated Reasoning using Possibilistic Logic: Semantics, Belief Revision and Variable Certainty Weights. In *IEEE Transactions on Knowledge and Data Engineering,* Vol.5, No.1, Feb. 1994, pp.64-71.

[9] R.C.T.Lee, "Fuzzy Logic and the Resolution Principle", *Journal of the ACM*, 19, 1972, pp.109-119.

[10] *Logic Programming and Soft Computing* - Uncertainty Theory in Artificial Intelligence Series, Research Studies Press, England, 1998.

[11] M.Mukaidono 82. Fuzzy Inference of Resolution Style. In *Fuzzy Sets and Possibility Theory*, Ed. R.R.Yager, Pergamon Press, New York, 1982, pp.224-231.

[12] J.A:Robinson. A Machine Oriented Logic Based on the Resolution Priciple. *J.ACM*, 12, 1, 1965, pp:23-41.

[13] Z.Shen, L.Ding and M.Mukaidono. A Theoretical Framework of Fuzzy Prolog Machine. *Fuzzy Computer*, Edt. M.M.Gupta, T.Yamakawa, North-Holland, Amsterdam, 1988.

[14] H.Virtanen. A Study in Fuzzy Logic Programming. In *Proceedings of the 12th European meeting on Cybernetics and Systems'94*, pp. :249–256, Austria, April 1994.

[15] T.J.Weigert, J.Tsai and X.Liu. Fuzzy Operator Logic and Fuzzy Resolution. *Journal of Automated Reasoning*, 10:59-78, 1993.

[16] H.Y.Yasui, Y.Hamada, M.Mukaidono. Fuzzy prolog based on Lukasiewicz implication and bounded product. In *Logic Programming and Soft Computing*, Research Studies Press, Uncertain Theory in Artificial Intelligence Series, 1998.