A Fuzzy Logic Programming Language

Francesca Arcelli

DIIIE Università di Salerno 84084 Fisciano (SA), Italy.

email: arcelli@ponza.dia.unisa.it

Ferrante Formato

DIIMA Università di Salerno 84084 Fisciano (SA), Italy.

email: formato@ponza.dia.unisa.it

May 21, 1997

Abstract

We present a new fuzzy logic programming language, called LIKELOG, to be used for approximated reasoning and for fuzzy deductive database applications. The core component of the system is the algorithm of unification, which expands the Martelli-Montanari unification algorithm introducing two similarity relations. The resolution mechanism is a straightforward application of this kind of expanded unification. Given such a fuzzy resolution rule we present a fuzzy logic programming system where computed answers are supplied with a degree of derivation. The LIKELOG system is more flexible respect to the traditional information systems. LIKELOG retrieves answers that are "similar" to the user query. In LIKELOG the user can not only insert queries, but can also introduce his/her own relation of similarity, in order to obtain answers that, although approximate, meet his/her own particular preferences.

1 Introduction

Logic Programming has proved to be particularly useful in several areas of artificial intelligence. Nevertheless, it is not well suited to model uncertain knowledge and approximated reasoning. For this purpose, since Zadeh ([12]) set the foundations of fuzzy set

theory, several extensions of logic programming based on fuzzy logic have been proposed in literature, as for example [5],[7], [4] ([3]), [11], [10].

If we want to fuzzify logic programming, we have to decide which parts or aspects of logic programming we are more interested to fuzzify. As stated in [11] we can use fuzzy logic programming to perform fuzzy inference, to describe fuzzy systems or to solve fuzzy problems. In our approach fuzzification is introduced through a suitable fuzzy closure operator that transforms a classical definite program into a fuzzy subset of the language of definite Horn clauses. A query is matched against a definite program clause according to an extended resolution rule that extends to the fuzzy context the classical resolution rule introduced by Robinson. As in the classical case, fuzzy resolution lies on fuzzy unification, seen as an extension of the classical unification algorithm of Martelli and Montanari ([6]). The fuzzy unification is based on similarity relations. Fuzzy resolution brings to the immediate definition of successfull derivation chains and computed answers. These are accompained by a degree of derivability specifying the degree of "acceptability" of the substitution involved by the unification process.

We describe how the inferential engine of our fuzzy logic programming language LIKELOG can be used in a knowledge-based system capable of dealing with approximate reasoning. The interpreter of the language has been written in Prolog and runs on PC-based architecture.

The paper is organized through the following sections. In section 2 we introduce the syntax of LIKELOG and the fuzzy similarity relations on which the language is based. In section 3 we introduce and define the fuzzy unification used in the derivation process. In section 4 we define the fuzzy resolution rules used by the language and we give an example of a LIKELOG session. Finally we conclude and briefly describe some future developments.

2 A Fuzzy Logic programming Language

We consider a first-order language $\mathcal{L}$ with sets of funnction symbols $\mathcal{C}$ and set of predicate symbols $\mathcal{R}$. The set of function symbols $\mathcal{C}$ has only 0-arity operators. Let X be a finite set of variable We call *set of expanded terms* the set $\mathcal{T}_{X,\mathcal{C}} = X \cup \mathcal{P}(\mathcal{C})$ where $\mathcal{P}(\mathcal{C})$ is the powerset of $\mathcal{C}$. We identify a constant $d \in \mathcal{C}$ with the singleton sets $\{d\}$ in $\mathcal{P}(\mathcal{C})$. We call

set of expanded closed terms the set $\mathcal{T}_C = \mathcal{P}(\mathcal{C})$.

To each element p in $\mathcal{R}$ we associate an arity function $\alpha(p)$ defined as usual. Given the expanded terms $t_1, \ldots, t_n \in \mathcal{T}_{X,C}$ and a constant $p \in \mathcal{R}$ such that $\alpha(p) = n$, we call *expanded 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 *ground atomic formula*. The terms and the predicates will be called *expanded first-order expressions.*

The fuzzy logic programming language, called LIKELOG, which we define is based on the language of Horn clauses, whose set of function symbols is $\mathcal{F}$ and the set of predicate symbols is $\mathcal{C}$. Remark that in the set of function symbols only constants occurr. This is because we aim at using our logic programming language within the context of deductive databases.

We adopt for LIKELOG the same syntax used in conventional logic programming ([8]).

As any other classical logic programming language, the language LIKELOG is based on the resolution rule of inference and on the unification process used along the resolution steps. The fuzzy unification process used in LIKELOG and the fuzzy resolution are both based on fuzzy similarity relations.

Similarity Relations

2.1 Similarity

The notion of similarity extends in the context of a multivalued logic of the classical notion of a equivalence 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)* $\nabla(a, a) = 1$ *for any* $a \in \mathcal{U}$
- *ii)* $\nabla(a, b) \geq \nabla(b, a)$ *for any pair of elements* $a, b \in \mathcal{U}$
- *iii)* $\nabla(a, c) \geq \nabla(a, b) \wedge \nabla(b, c)$ *for any* $a, b, c \in \mathcal{U}$

Given a similarity relation ∇ on $\mathcal{U}$, we can associate to every set of elements of $\mathcal{U}$ a degree expressing the extent to which a set X does not contain 'dissimilar' elements. For this reason we call *clouds* the elements of the powerset of $\mathcal{P}(\mathcal{U})$.

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

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

$$\mu_\nabla(\emptyset) = 1$$

We have the following property:

Lemma 1 *Assume that $X \neq \emptyset$ and that $c \in X$. Then*

$$\mu_\nabla(X) = \bigwedge_{x \in X} \nabla(x,c)$$

We assume as primitives two similarity relations $eq_\mathcal{R}$ and $eq_\mathcal{C}$ on $\mathcal{R}$ and $\mathcal{C}$, respectively, where $eq_\mathcal{R}(f,g) = 0$ if $\alpha(f) \neq \alpha(g)$ for any $f, g \in \mathcal{R}$. We use $eq_\mathcal{C}$, $eq_\mathcal{R}$, $\mu_{eq_\mathcal{R}}$ and $\mu_{eq_\mathcal{C}}$ to define a fuzzy relation eq on the set of terms $\mathcal{T}_{X,\mathcal{C}}$ as follows:

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

First, we extend this relation to the set of predicates:

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

This relation is used to define a fuzzy program associated to a given program in LIKELOG. Given a set of Horn clauses $\mathcal{CL}$ we call $\mathcal{CL}'$ the set composed by the set of ground facts and the set of rules whose head does not contain constants. The clauses in $\mathcal{CL}'$ do not arise problems in defining the similarity relation, since they do not use constants in their heads.

Definition 3 *A LIKELOG program, or more simply a program $\mathcal{P}$, is a subset of $\mathcal{CL}'$.*

This restriction does not affect the expressivity of the language, since every Horn clause with constants as argument in the head is logically equivalent to a clause in $\mathcal{CL}'$ and according to our interest to use LIKELOG for deductive database applications, it is reasonable for a deductive database to have such clauses in its set of rules.

Second, we extend the fuzzy relation eq to the set $\mathcal{CL}'$ by setting a new relation eq^* as follows:

$$\begin{aligned} eq^*(a,b) &= eq(a,b) \text{ if } \alpha(a) = \alpha(b). \\ eq^*(p:-q_1,\ldots,q_n,p':-q'_1,\ldots,q'_n)) &= eq(p,q) \\ 0 \text{ otherwise} & \end{aligned}$$

Given a program $\mathcal{P}$, we define a fuzzy subset p of $\mathcal{CL}'$ that turns to be the *softening* of program $\mathcal{P}$ according to the similarity relation eq^*. To do this, we set $p = Sim(\mathcal{P})$ where Sim is an operator from $P(\mathcal{CL}')$ into $P(\mathcal{CL}')$ which is defined as follows:

$$Sim(\mathcal{P})(p) = \sup_{\alpha' \in \mathcal{P}} \{eq^*(\alpha', p)\} \tag{1}$$

Given a program $\mathcal{P}$, we call *fuzzy program associated to* $\mathcal{P}$ the fuzzy subset $Sim(\mathcal{P})$.

3 A Fuzzy Unification Algorithm

Following the idea of the Martelli-Montanari algorithm for first-order term unification, (see [6]) we propose an algorithm to find a unifier of a set of term equations and, at the same time, the "degree of unification" of this unifier.

The main difference between this algorithm and the classical unification algorithm of Martelli-Montanari stands in the way ground equations are dealt with. In the Martelli-Montanari unification procedure, only identical ground equations of the form $a = a$ are eliminated, otherwise unification fails.

The unifier is computed using a particular data-structure, the 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 $\{x_1, x_2, \ldots, x_n\} = (a_1, a_2, \ldots, a_m)$.

We describe an algorithm taking a set of term equations on $\mathcal{T}_{X,\mathcal{C}}$ as input and giving as output a set of clouds and an "extended" unifier θ, i.e. a unifier whose range is the set $\mathcal{T}_{X,\mathcal{C}}$. The degree of unifiability of the unifier θ, written as $\nu(\theta)$, is assumed as the infimum among the eq-singleton measure of the set of clouds (see section 2).

In our algorithm the clouds are originated by a term equation of the kind $a = b$ where a and b are constants in $\mathcal{C}$; mismatches of this kind are caused either by the term equations in input, or by the links produced by the variables. While in this case classical unification

does not provide solutions, in our approach a cloud of the form $\{a, b\}$ is created. Such cloud 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 by constants that, if equal, would give a classical solution. For this reason, the degree of unifiability $\nu(\theta)$ can be interpreted as the degree to 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 generalized unification algorithm is applied to a system of equations S over the set of first-order expressions. Such equations are converted in a set $S' = \{X_1 = D_1, \ldots X_t = D_t\}$ where X_i and D_i are (possibly empty) sets of variables and clouds, respectively. The set S' has a straightforward implicit generalized unifier: for any generalized equation of the kind $X = D$, simply take $\{x \rightarrow D\}$ for any variable $x \in X$. If $D = \emptyset$, the generalized unifier is simply obtained by linking the variables in X one another. The generalized unifier θ is associated to a "degree of unifiability", given by $\nu(S) = \bigwedge_{i=1}^{t} \mu(D_i)$. Furtherly, we extend the generalized unifier by trying to expand the clouds without altering the cost payed for unification,i.e. the degree $\nu(S)$. 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 unifiability of its implicit generalized unifier. Such an operation is called *algebraic lifting*. A procedural description of the fuzzy unification algorithm is given below:

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

procedure mgu1(Z, θ, 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; θ := 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', 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 equation of the kind $p = q$ where p and q are predicates with the same arity;

Convert(S) is a function whose output is a system of multiequations: this function 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.

Compact(Z) is a function that takes a system of multiequations Z as input and yields a "compactified" version of Z; given equivalence closure $\bar{\mathcal{R}}$ of the relation $\mathcal{R}$ on the set of multiequations, such that $X = M\mathcal{R}X' = M'$ iff $X \cap X' \neq \emptyset$ or $M \cap M' \neq \emptyset$, *Compact*(Z) yields a system of multiequations obtained by merging, member to member, the multiequations in the same classes of equivalence *Solve*(S) is a function that takes a triangular set of multiequations as argument and derives an "extended" substitution as follows, where M is a cloud of constants in $\mathcal{P}(\mathcal{C})$:

$$\begin{aligned} solve(\{x_1, \ldots, x_n\} = M) &= \{x_1 \rightarrow M, \ldots, x_n \rightarrow M\} \\ solve(\{x_1, \ldots, x_n\} = \emptyset) &= \{x_1 \rightarrow x_j, \ldots, x_{n-1} \rightarrow x_j\} \end{aligned}$$

For some integer j in $[1..n]$.

Finally, clouds are extended keeping constant the cost one must pay for unification i.e. the minimum of *eq*-singleton measure of all the clouds of the final system. We add to each cloud the elements in $\mathcal{C}$ and $F\mathcal{R}$ whose similarity degree is greater than the minimum of *eq*-singleton-measures of the clouds in the system. This is a one-step process, since after the first application of such method the clouds remain the same. The extension of the clouds is obtained through the application of the operator H_λ from $\mathcal{P}(\mathcal{C})$ into itself , defined as follows: $H_\lambda(D) = \{y \in L | eq(x, y) \geq \lambda \text{ for some } x \in D\}$.

The extended unification algorithm is implemented in Prolog with a concatenation of predicates expressing the primitive transformations described above: the predicate `solve` takes a system of equations as input and produces a solved system as output. The predicate `split` takes a system of equation and applies the primitive *Split*. The resulting system is given as input to the predicate `convert_sys`, that implements the primitive *convert*. The resulting system is then compacted through the predicate `compact_sys`. The output system of this latter predicate goes through the algebraic lifting, performed by the predicate `exp_alg_sys`. Finally the resulting system is again compactified.

```
solve(SysIn,SysOut) :-
split(SysIn,SysSplitted),
convert_sys(SysSplitted,SysConverted),
compact_sys(SysConverted,SysCompactified),
compute_lambda(SysCompactified,Lambda),
exp_alg_sys(SysCompactified,SysExp,Lambda),
compact_sys(SysExp,SysOut),!.
```

4 Fuzzy Resolution

We extend the classical resolution rule to the fuzzy environment and we define a fuzzy resolution rule based on the similarity relation *eq* and on the fuzzy unification algorithm defined in the previous section.

Definition 4 *Given a goal formula* $\leftarrow G_1 \wedge \ldots \wedge G_i \wedge \ldots \wedge G_n$ *and a definite clause* $A \leftarrow B_1 \wedge \ldots \wedge B_m$, *we set the following fuzzy resolution rule:*

$$\frac{\leftarrow G_1 \wedge \ldots \wedge G_i \wedge \ldots \wedge G_n \; A \leftarrow B_1 \wedge \ldots \wedge B_m}{\theta(G_1 \wedge \ldots G_{i-1}, \wedge B_1 \wedge \ldots \wedge B_m \wedge G_{i+1} \wedge \ldots \wedge G_n)} \; N \tag{2}$$

where θ *is a unifier and* N *is a set of clouds derived from the fuzzy unification algorithm with* $\{A = G_i\}$ *as input.*

Definition 5 *Let* G *be a goal clause and let* $\mathcal{P}$ *be a LIKELOG program, a* refutation *of* G *with respect to the program* $\mathcal{P}$ *is a finite sequence of resolvents* $R_1, R_2, \ldots, R_n$ *such that* $R_n = []$ *and* R_{i+1} *is derived from* R_i *by fuzzy resolution rule, for all* $i = 1, \ldots, n$.

A *computed answer* associated to a successful chain of resolution steps $\mathcal{R} = (R_i)_{i \in [1,n]}$ is a substitution $\theta = \theta_1 \circ \theta_2 \circ \ldots \circ \theta_{n-1}$ where θ_i is a unifier derived at the i-th step of resolution. Naturally, the composition of the single unifier θ_i will be computed by eventually renaming apart the clashing free variables, as usual in classical resolution.

A refutation of a goal G with respect to a program $\mathcal{P}$ produces two sets of clouds, one is given by $\{N_1, N_2, \ldots, N_r\}$ through the resolution steps, and the other is given by a set of clouds obtained by the computed answer $\{N'_1, N'_2, \ldots N'_s\}$. If we denote with Z the normalized and algebrically extended form of the system of term equations: $\{\emptyset = N_1, \ldots, \emptyset = N_r, \emptyset = N'_1, \ldots, \emptyset = N'_s\}$, the number $\nu(Z)$ is called *degree of derivability* of G with respect to $\mathcal{P}$ and to the chain $\mathcal{R} = (R_i)_{i \in [1,n]}$.

Definition 6 *Given a goal G and a program $\mathcal{P}$, the* degree of derivability *of G from $\mathcal{P}$, $D(G,\mathcal{P})$, is defined by the expression*

$$D(G,\mathcal{P}) = \sup R(G,\mathcal{P})$$

where $R(G,\mathcal{P})$ is the set of the degrees of derivability of G with respect to some successful derivation chain $\mathcal{R} = (R_i)_{i\in[1,n]}$ in $\mathcal{P}$ such that $R_1 = G$.

A fuzzy resolution chain, i.e. a succesful sequence of resolvents, is easily computed through a Prolog program: the main part of the program, containing the predicate `resolution_ chain` is illustrated below. It is an iterative loop controlled by the first argument of the predicate. When this argument is instantiated with the empty list, the resolvent is empty, hence the resolution process stops. Otherwise, a single resolution step is performed (`resolve_step`) and the resulting mgu and list of clouds are added to the list of substitutions and to the list of clouds, respectively. Finally, the predicate `resolution_chain` is called again, with a new resolvent and a new value for the list of substitutions and the list of clouds. One of the main advantage offered by the implementation of the fuzzy resolution in Prolog is that the backtracking mechanism is given for free by the backtracking mechanism built-in in the meta-language.

```
resolution_chain([],Clouds,Clouds,Comput_answer,Comput_answer) :- !.
/* empty clause: success achieved! */

resolution_chain(Resolvent,Clouds,Acc,Unifier,Acc2) :-
  resolve_step(Resolvent,NewResolvent,Mgu),
 /*  resolution step */
compose(Mgu,Unifier,New_unifier),
/* update the computed answer */
cloud_extraction_step(Resolvent,List1),
 /* update the list of  clouds */
append(Clouds,List1,New_list_ofclouds),
resolution_chain(NewResolvent,New_list_of_clouds,New_unifier,Acc1,Acc). /* redo
```

4.1 An Example of a LIKELOG session

Let $\mathcal{P}$ be a LIKELOG program containing a data-base with a list of books titles classified according to their subject and two relations **exciting_to_read** and **cheap** on the interest expectations and on the cost convenience of the books respectively.

```
adventurous(sandokan).
 adventurous(gulliver).
 horror(dracula).
 horror(frankenstein).
 thriller(murder on the Orient Express).
 thriller(james bond).

exciting_to_read(X) :- adventurous(X).

/*  LIKELOG Program */

eq(adventurous,horror) = ?
>0.6

eq(adventurous,thriller) = ?
>0.8

/* Relation eq defined by the user  */

eq(adventurous,horror)=0.6
eq(adventurous,thriller)=0.8
eq(adventurous,adventurous)=1
eq(horror,adventurous)=0.6
eq(horror,thriller)=0.6
eq(horror,horror)=1
eq(thriller,horror)=0.6
eq(thriller,adventurous)=0.8
eq(thriller,thriller)=1
```

```
/* The minimum similarity relation containing eq automatically generated */

?-exciting_to_read(X).

/* Existential  query  by the user */

?- X --> sandokan, degree = 1
?- X --> gulliver, degree = 1
?- X --> murder on the Orient Express, degree = 0.8
?- X --> jamesw bond, degree = 0.8
?- X --> dracula, degree = 0.6
?- X --> frankenstein, degree = 0.6
```

Like in classical logic programs, the operational semantics lies on the softening of the concept of 'derivation', essentially based on the resolution mechanism, that is extensionally represented by the set of ground clauses that are provable by a program. The declarative semantics of a program $\mathcal{P}$ is computed by extending the classical least fixed point operator $T_{\mathcal{P}}$ to a fuzzy closure operator. The least fixed point of such operator will be assumed as the declarative semantics of the fuzzy logic program.

The complete semantics, both the declarative and the operational semantics and the proof of the coincidence of the two semantics and the stability of them respect to the similarity relation are formally given in [2].

5 Conclusions and Future Developments

We have described a new fuzzy logic programming language, called LIKELOG based on similarity relations. As in the classical case, the core of the system is an algorithm of unification. In our approach, this algorithm is an extension of the classical Martelli-Montanari unification algorithm by means of an unification index stemming from a relation of similarity among the elements of the domain. A fuzzy rule of resolution has been defined based on the fuzzy unification algorithm. Several problems have to be faced, in order

to achieve a better implementation, such as the refinement of the computational model and the automatic generation of the minimum similarity relation, given a set of basic membership values supplied by the users

References

[1] F.Arcelli, F.Formato and G.Gerla, Similitude-Based Unification as a Foundation of Fuzzy Logic Programming , *Proceedings of Internat. Workshop on Logic Programming and Soft Computing*, Bonn, sept.1996.

[2] F.Arcelli, F.Formato and G.Gerla, A Semantic of a Fuzzy Logic Programming Language, Tech.Report, University of Salerno, DIIIE 2-97, 1997.

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

[4] H.Kikuchi and M.Mukaidono, PROFIL - Fuzzy Interval Logic Prolog, *Preprints of Internation workshop on Fuzzy Systems Appplications*, pp.:205-206, 1988.

[5] M.Ishizuka and N.Kanai, "Prolog -ELF Incorporating Fuzzy Logic", in *Proceedings of 9th IJCAI*, 1985, pp.::701-703.

[6] A. Martelli and U.Montanari, "An Efficient Unification Algorithm", *ACM Transactions on Programming Languages and Systems*, Vol 4, No 2, April 1982.

[7] M.Mukaidono, Z.Shen and L.Ding, "Fundamentals of Fuzzy Prolog", *Int.Jornal of Approximated Reasoning*, 3,2, 1989.

[8] L:Sterling and E.Shapiro, *The Art of Prolog, MIT PRESS, 1987.*

[9] J.A.Robinson, A Machine-oriented logic based on resolution principle, in *JACM*, 12(1), pp.23 41, Jan. 1983.

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

[11] 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.

[12] L.Zadeh, "Fuzzy Sets", *Information and Control*, 8, 1965.