Journal of Universal Computer Science, vol. 12, no. 9 (2006), 1087-1103
submitted: 31/12/05, accepted: 12/5/06, appeared: 28/9/06 © J.UCS

Creation and Evaluation of Fuzzy Knowledge-base

Ágnes Achs
(University of Pécs, Hungary
achs@witch.pmmf.hu)

Abstract: In this paper we give a possible model for handling uncertain information. The concept of fuzzy knowledge-base will be defined as a quadruple of any background knowledge, defined by the proximity of predicates and terms; a deduction mechanism: a fuzzy Datalog program; a connecting algorithm, which connects the background knowledge with the program and a decoding set of the program, which help us to determine the uncertainty level of the results. Evaluation strategies will also be presented.

Keywords: fuzzy Datalog, fuzzy knowledge-base, proximity, evaluation strategies
Categories: I.2.3

1 Introduction

A large part of human knowledge can not be modelled by pure inference systems, because this knowledge is often ambiguous, incomplete and vague. The study of inference systems is faced in literature with several and often very different approaches. When knowledge is represented as a set of facts and rules, this uncertainty can be handled by means of fuzzy logic. The concept of deductive databases and fuzzy logic is discussed in the classical works, for example in [CGT], [DP], [L], [N], [U].

A few years ago in [AK1] and [A1] a possible combination of Datalog-like languages and fuzzy logic was given. In these works the concept of fuzzy Datalog was introduced by completing the Datalog-rules and facts by an uncertainty level and an implication operator. In [AK2] we extended the fuzzy Datalog to fuzzy relational databases.

Parallel with these works, there were researches on possible combinations of Prolog language and fuzzy logic. Several solutions were suggested for this problem. These solutions propose different methods for handling uncertainty. Most of them use the concept of similarity, but in various ways. They consider similarity as reflexive, symmetric and transitive relation. Some of them take a classification according to similarities and use these equivalence classes to make unification and fuzzy resolution, for example [AFG], [S]. In [V1], [V2] the author deals with linguistic variables and their linguistic values, and gives a definition of the fuzzy unification algorithm by means of these values. In [SS] the authors discuss a special kind of fuzzy unification. They deal with the problem of missing parameters and mismatching predicates and parameters. They define the edit distance of terms, which is compound according to the number of mismatches, and give the unification algorithm according to these distances.

In this paper, continuing our former concept, we give another possible model for handling uncertain information, based on the extension of fuzzy Datalog. We will be also supported by the concept of similarity, but it is not expected to be transitive.

Although in [AK1], [AK2], [AK3] the authors deal with the concept of fuzzy Datalog in detail, in the following we give a short summary of it.

2 The fuzzy Datalog

A Datalog program consists of facts and rules. In fuzzy Datalog we can complete the facts by an uncertainty level, the rules by an uncertainty level and an implication operator, which means, that evaluating the fuzzy implication connected to the rule, its truth-value according to the implication operator is at least the uncertainty level of the rule. According to this operator we can compute the uncertainty level of the rule-head.

For example, if in the program

```
beautiful('Mary'); 0.7.
likes('John', X) ← beautiful(X); 0.8; I.
```

the implication operator is the Gödelian, (that is $I(\alpha,\beta) = 1$, if $\alpha \leq \beta$, $I(\alpha,\beta) = \beta$ otherwise), then the uncertainty level of the rule-head is the minimum of the uncertainty levels of the rule-body and the rule. In our case this means, that

```
likes('John','Mary'); 0.7,
```

that is, John likes Mary at least 0.7 level.

Now we are going to summarise the concept of fuzzy Datalog (fDATALOG) based on [AK1], [AK2].

Definition 1.
A fDATALOG rule is a triplet r;β;I, where r is a formula of the form

$$A \leftarrow A_1,...,A_n \qquad (n \geq 0).$$

A is an atom (the head of the rule), $A_1,...,A_n$ are literals (the body of the rule); I is an implication operator and $\beta \in (0,1]$ (the level of the rule).

For getting finite result, all the rules in the program must be safe. A fDATALOG rule is safe if

– all variables occurring in the head also occur in the body;
– all variables occurring in a negative literal also occur in a positive literal.

A fDATALOG program is a finite set of safe fDATALOG rules. There is a special type of rule, called fact. A fact has the form $A \leftarrow ;\beta;I$. Further on we refer to facts as (A,β), because according to implication I, the level of A easily can be computed.

The semantics of fDATALOG is defined as the fixpoints of consequence transformations. Depending on these transformations we can define two semantics for fDATALOG. Deterministic semantics is the least fixpoint of deterministic transformation, nondeterministic semantics is the least fixpoint of nondeterministic transformation. According to the deterministic semantics, the rules of a program are evaluated parallel, while in the nondeterministic case the rules are considered independently one after another. Further on we deal only with nondeterministic

transformation, because we can't use the deterministic semantics when the program has any negation. This transformation is the following:

Definition 2.
Let B_P the Herbrand base of the program P, and let $F(B_P)$ denote the set of all fuzzy sets over B_P. The consequence transformation NT_P: $F(B_P) \rightarrow F(B_P)$ is defined as

$$NT_P(X) = \{(A, \alpha_A)\} \cup X,$$

where

$$(A \leftarrow A_1,\ldots,A_n ; I ; \beta) \in \textit{ground}(P),$$
$$(|A_i|,\alpha_{A_i}) \in X,\ 1 \le i \le n,\ \alpha_A = \max(0,\min\{\gamma \mid I(\alpha_{body},\gamma) \ge \beta\}).$$

ground(P) denotes the set of all ground instances of P's rules, $|A_i|$ denotes the kernel of the literal A_i, (that is it is the ground atom A_i, if A_i is a positive literal, and $\neg A_i$, if A_i is negative).

In [AK2] it was proved, that starting from the set of facts, both deterministic and nondeterministic transformations have their own fixpoint, which is the least fixpoint in the case of negation-free program. The fixpoint of nondeterministic transformation is denoted by $lfp(NT_P)$.

It was also proved, that $lfp(NT_P)$ is a model of P, so $lfp(NT_P)$ can be defined as the nondeterministic semantics of the fDATALOG program P. For function- and negation-free fDATALOG, the two semantics are the same, but as it was mentioned above, the deterministic semantics is not suitable when the program has any negation. In this case the nondeterministic semantics is applicable under certain conditions. This condition is the stratification. Stratification gives an evaluating sequence in which the negative literals are evaluated first. (Detailed in [AK2].)

Example 1.
Given the next fDATALOG program:

```
1.    (r(a), 0.8).
2.    p(x) ← r(x),¬q(x); 0.6; I.
3.    q(x) ← r(x); 0.5; I.
4.    p(x) ← q(x); 0.8; I.
```

The stratification is: $P_1 = \{r,q\}$, $P_2 = \{p\}$, so the evaluation order is: 1., 3., 2., 4.
(Precisely: firstly 1. and 3. in arbitrary order, then 2. and 4. in arbitrary order.)
Then in the case of Gödelian implication operator the nondeterministic semantics of the program is $lfp(NT_P) = \{(r(a),0.8); (p(a),0.5); (q(a),0.5)\}$.

Further on we deal only with nondeterministic semantics.

3 Background knowledge

The facts and rules of a fDATALOG program can be regarded as any kind of knowledge, but sometimes we need other information, some background knowledge, in order to get answer for a query. In this section we give a possible model of background knowledge, which is based on the concept of proximity.

Definition 3.
A proximity on a domain D is a fuzzy subset S_D: D×D → [0,1] such that the following properties hold:

$S_D(x,x) = 1$ for any $x \in D$ (reflexivity)
$S_D(x,y) = S_D(y,x)$ for any $x,y \in D$ (symmetry).

If a proximity is transitive, that is

$S_D(x,z) \geq \min(S_D(x,y), S_D(y,z))$ for any $x,y,z \in D$,

then it is called similarity.

Example 2.
Let us consider the next proximity matrix:

	a	b	c	d	e
a	1	0.7	0.8	0.7	0.8
b	0.7	1	0.7	0.9	0.7
c	0.8	0.7	1	0.7	0.8
d	0.7	0.9	0.7	1	0.7
e	0.8	0.7	0.8	0.7	1

It can be easily checked that the proximity defined by this matrix is transitive, so it is similarity.

In our model the background knowledge is a set of proximity sets:

Definition 4.
Let $d \in D$ be an element of domain D. The proximity set of d is a fuzzy subset over D:

$$S_d = \{(d_1, \lambda_1), (d_2, \lambda_2), \ldots, (d_n, \lambda_n)\},$$

where $d_i \in D$ and $S_D(d, d_i) = \lambda_i$ for i = 1,…n.

Based on proximities we can construct the background knowledge, which gives information about the proximity of terms and predicate symbols.

Definition 5.
Let C be a set of ground terms, R be a set of predicate symbols. Let SC and SR be proximities over C and R respectively. The background knowledge is the union of proximity sets of C and R:

$$Bk = \{SC_t \mid t \in C\} \cup \{SR_p \mid p \in R\}$$

4 Fuzzy knowledge-base

We made two steps on the way leading to the concept of fuzzy knowledge-base: we defined the concept of fuzzy Datalog program and the concept of background knowledge. The next step is the connection of these concepts which is made by a connecting algorithm. This algorithm gives a modified fDATALOG program, which

is the extension of the original one according to proximity. Evaluating this program, the resulting uncertainty levels are not yet the final ones; those can be computed from these levels and from the proximity values of actual predicates and theirs arguments. To do this, we need the concept of decoding functions. It is expectable, that in the case of identity the decoding functions should not change the original level, but in other cases the final level should be less or equal than the original level or than the proximity values. Furthermore we require the decoding function to be monotone increasing.

Definition 6.
A decoding function is an (n+2)-ary function :

$$\varphi(\alpha, \lambda, \lambda_1, \ldots, \lambda_n) : (0,1] \times (0,1] \times (0,1] \times \ldots \times (0,1] \rightarrow [0,1],$$

so that

$$\varphi(\alpha, \lambda, \lambda_1, \ldots, \lambda_n) \leq \min(\alpha, \lambda, \lambda_1, \ldots, \lambda_n),$$
$$\varphi(\alpha, 1, 1, \ldots, 1) = \alpha, \text{ and}$$
$$\varphi(\alpha, \lambda, \lambda_1, \ldots, \lambda_n) \text{ is monotone increasing in all arguments.}$$

Example 3.

$$\varphi_1(\alpha, \lambda, \lambda_1, \ldots, \lambda_n) = \min(\alpha, \lambda, \lambda_1, \ldots, \lambda_n);$$
$$\varphi_2(\alpha, \lambda, \lambda_1, \ldots, \lambda_n) = \min(\alpha, \lambda, (\lambda_1 \cdot \cdot \cdot \lambda_n));$$
$$\varphi_3(\alpha, \lambda, \lambda_1, \ldots, \lambda_n) = \alpha \cdot \lambda \cdot \lambda_1 \cdot \cdot \cdot \lambda_n$$

are decoding functions.

It is worth mentioning that any triangular norm is suitable for decoding function, for example the above min and product operators are t-norms.

We have to assign decoding functions to all – but at least to the head – predicates of the program. The set of decoding functions will be the decoding set of the program. To define this set, we need the concept of the functor, which is characterized by the predicate symbol and the arity of an atom, that is for example in the case of $p(t_1, t_2, \ldots, t_n)$, the functor is p/n.

Definition 7.
Let P be a fuzzy Datalog program, and F_P be the set of the program's functors. The decoding set of P is:

$$\Phi_P = \{ \varphi_q(\alpha, \lambda, \lambda_1, \ldots, \lambda_n) \mid \forall q/n \in F_P \}$$

Clearing the required concepts, we can define the fuzzy knowledge-base:

Definition 8.
A fuzzy knowledge-base (fKB) is a quadruple (Bk, P, Φ_P, mA), where Bk is a background knowledge, P is a fuzzy Datalog program, Φ_P is a decoding set of P and mA is any modifying (or connecting) algorithm.

Definition 9.
Let (Bk, P, Φ_P, mA) be a fuzzy knowledge-base, and mP be the modified program according to mA. The consequence of the knowledge-base, denoted by C(Bk, P, Φ_P, mA), is the least fixpoint of mP: C(Bk, P, Φ_P, mA) = lfp(mP).

5 Modifying algorithms

There may be several modifying algorithms, but in the following we define only two kinds of them. According to the first one we modify the program and use the original consequence transformation for evaluation; in the second case a modified consequence transformation is applied for the original program. Here we summarize these methods.

5.1 Simple modification (algorithm mA1)

Let P be a fuzzy Datalog program, and let Bk be any background knowledge. By the proximities of the background knowledge we can define the modified fDATALOG program, mP: Let us replace each predicate symbol p of the program P by SR_p, each ground term $t \in H_p$ by SC_t and each variable x by $X=\{x\}$. (Note: it may occur, that SR_p or SC_t is not in Bk, in this case $SR_p = \{(p,1)\}$ or $SC_t = \{(t,1)\}$.)
The algorithm of modification is the following:

Algorithm 1.

```
Procedure modification(P, mP)
        mP := ∅
        while not(empty(P)) do
                (r;I;β) := first rule of P
                (R;I;β) := (replace(r);β;I)
                mP := mP U (R;β;I)
                P := P – {(r;β;I)}
        endwhile
endprocedure

function replace(r)
        Pred_r := set of r's predicate symbols
        Term_r := set of r's ground terms
        Var_r := set of r's variables

        while not(empty(Pred_r)) do
                q := first predicate-symbol of Pred_r
                Q := SR_q
                Pred_r := Pred_r – {q}
        endwhile
        while not(empty(Term_r)) do
                t := first ground term of Term_r
                T := SC_t
                Term_r := Term_r – {t}
        endwhile
        while not(empty(Var_r)) do
                x := first variable of Var_r
```

```
            X := {x}
            Var_r := Var_r – {x}
        endwhile
        return replace(r)
endfunction
```

The modified fDATALOG program, mP is evaluable as an ordinary fDATALOG program. The resulting fixpoint contains the proximity sets, from which we have to decide the final ground terms. These terms form the fixpoint of modified program:

Definition 10.
The least fixpoint of the modified program, mP is:

$$lfp(mP) = \bigcup \{(q(t_1, t_2 , \ldots, t_n); \varphi_q (\alpha_q, \lambda_q, \lambda_{t_1} ,\ldots, \lambda_{t_n})) \mid$$
$$\forall(Q((T_1, T_2 , \ldots, T_n); \alpha) \in lfp(NT_{mP}),\ (q , \lambda_q) \in Q, (t_i , \lambda_{t_i}) \in T_i , 1 \le i \le n\}.$$

Because in the case of $(q(t_1, t_2,\ldots t_n); \alpha) \in lfp(NT_P)$

$$(Q(T_1,T_2,\ldots,T_n); \alpha) = (SR_q (SC_{t_1},SC_{t_2},\ldots,SC_{t_n}); \alpha) \in lfp(NT_{mP}),$$

and $\varphi_q(\alpha,1,1,\ldots,1) = \alpha$, therefore the next proposition is true :

Proposition 1.

$$lfp(NT_P) \subseteq C(P, Bk, \Phi_P, mA1).$$

Example 4.
Let us see the fDATALOG program of the first example (with the Gödelian implication operator) and let us extend it by a background knowledge and a decoding set!

```
(r(a), 0.8).
p(x) ← r(x),¬q(x); 0.6; I.
q(x) ← r(x); 0.5;I.
p(x) ← q(x); 0.8;I.
```

	p	q	r	s	t
p	1	0.4			
q	0.4	1			
r			1	0.6	0.7
s			0.6	1	
t			0.7		1

$\varphi_p(\alpha, x, y) = \varphi_q(\alpha, x, y) = \min(\alpha, x, y);$
$\varphi_r(\alpha, x, y) = \alpha \cdot x \cdot y$

	a	b
a	1	0.8
b	0.8	1

According to example 1, $lfp(NT_{mP}) = \{(R(A),0.8); (P(A),0.5); (Q(A),0.5)\} =$
{({(r,1),(s,0.6),(t,0.7)}({(a,1),(b,0.8)}), 0.8); ({(p,1),(q,0.4)}({ (a,1),(b,0.8)}),0.5); ({(q,1),(p,0.4)}({ (a,1),(b,0.8)}), 0.5)}.

From this lfp(mP) = {(r(a),0.8), (r(b),0.64), (s(a),0.48), (s(b),0.384), (t(a),0.56), (t(b),0.448), (p(a),0.5), (p(b),0.5), (q(a),0.5), (q(b),0.5)}.

5.2 Transformation modification (algorithm mA2)

Previously we modified the program and used the original consequence transformation for evaluation; in the next case a modified consequence transformation is applied for the original program. The modified program is denoted by mP also.

The original consequence transformation is defined over the set of all fuzzy sets of P's Herbrand base, that is over $F(B_P)$. To define the modified transformation's domain, let us extend P's Herbrand universe with all possible ground terms occurring in background knowledge – so we get the modified Herbrand universe, mH_P. Let the modified Herbrand base, mB_P be the set of all possible ground atoms whose predicate symbols occur in $P \cup Bk$ and whose arguments are elements of mH_P. The modified consequence transformation is defined over this modified Herbrand base, so we can deduce to the atoms being in the rule-heads and the atoms being similar to them. More precisely:

Definition 11.
The modified consequence transformation $mNT_P : F(mB_P) \rightarrow F(mB_P)$ is defined as

$$mNT_P(X)=\{(q(s_1,...,s_n), \phi_p(\alpha, \lambda_q, \lambda_{s_1}, \ldots, \lambda_{s_n}) \mid (q, \lambda_q) \in SR_p;\ (s_i, \lambda_{s_i}) \in SC_{t_i},\ 1 \le i \le n\} \cup X,$$

where

$$(p(t_1,...,t_n) \leftarrow A_1,...,A_k ; \beta; I) \in \mathit{ground}(P),$$
$$(|A_i|, \alpha_{A_i}) \in X,\ 1 \le i \le k,\ \alpha = \max(0,\min\{\gamma \mid I(\alpha_{body}, \gamma) \ge \beta\}).$$

$|A_i|$ denotes the kernel of the literal A_i.

It can be proven, that starting from the facts of the program and creating the powers of the transformation mNT_P, finally we reach the fixpoint. As the modification has no influence on the order of rules, therefore neither it has influence on the stratification. So the modified consequence transformation has a least fixpoint in the case of stratified program too.

The fixpoint of the modified program, that is the consequence of knowledge-base is defined as the fixpoint of the transformation mNT_P.

Definition 12.
The least fixpoint of the modified program, mP is:

$$lfp(mP) = lfp(mNT_P).$$

From the above construction it is obvious:

Proposition 2.

$$lfp(NT_P) \subseteq C(P, Bk, \Phi_P, mA2).$$

Example 5.
Let us see again the first three component of the knowledge-base from example 4, but let us change the modifying algorithm to mA2!

The starting set of facts is X = {(r(a), 0.8)}, and the evaluation order is the same, as in the first example, that is: 1., 3., 2., 4.

According to the proximities the starting set turns into the following set:
X = {(r(a), 0.8), (r(b),0.64), (s(a),0.48), (s(b),0.384), (t(a),0.56), (t(b),0.448)}.

The third rule expands this set with the set {(q(a), 0.5), (q(b), 0.5)}, then according to proximity this set is expanded with the set {(p(a),0.4), (p(b),0.4)}.

According to the second rule, we can deduce for the atoms {(p(a), 0.5), (p(b), 0.5)}. As the next steps don't give new atoms, therefore the least fixpoint of the transformation and so the consequence of knowledge-base is:
lfp(mP) = {(r(a),0.8), (r(b),0.64), (s(a),0.48), (s(b),0.384), (t(a),0.56), (t(b),0.448), (q(a),0.5), (q(b),0.5), (p(a), 0.5), (p(b), 0.5)}

In our case the two algorithms led to the same consequence, but in general the consequence of the second knowledge-base is wider than the consequence of the first one. Comparing the two constructions it is obvious:

Proposition 3.

$$C(P, Bk, \Phi_P, mA1) \subseteq C(P, Bk, \Phi_P, mA2).$$

It is also easily provable, that in both cases lfp(mP) is a model of P, but $lfp(NT_P) \subseteq lfp(mP)$, so it is not a minimal one.

Example 6.
Let us see the next knowledge-base:

lo(x,y) ← gc(y), mu(x); 0.7; I.
(fv(V), 0.9).
(mf(M), 0.8).

$$I(\alpha, \beta) = \begin{cases} 1 & \text{if } \alpha \le \beta, \\ \beta & \text{otherwise} \end{cases}$$

	lo	li	gc	fv	mu	mf
lo	1	0.8				
li	0.8	1				
gc			1	0.75		
fv			0.75	1		
mu					1	0.6
mf					0.6	1

$\phi_{lo} := \phi = \phi(\alpha,x,y,z) := \min(\alpha,x,y,z)$
$\phi_{fv} := \theta = \theta(\alpha,x,y) := \alpha \cdot x \cdot y$
$\phi_{mf} := \omega = \omega(\alpha,x,y) := \min(\alpha, x \cdot y)$

(according to the modifying algorithm, it is enough to consider only the decoding functions of head-predicates)

	B	V	M
B	1	0.9	
V	0.9	1	
M			1

Then
C(P, Bk, Φ_P, mA1) = {(fv(V),0.9); (fv(B),0.81); (gc(V),0.675); (gc(B),0.6075); (mf(M),0.8); (mu(M),0.6)}

C(P, Bk, Φ_P, mA2) = {(fv(V),0.9); (gc(V),0.675); (fv(B),0.81); (gc(B),0.6075); (mf(M),0.8); (mu(M),0.6); (lo(M,V),0.6); (lo(M,B),0.6); (li(M,V),0.6); li(M,B),0.6)}.

Note: The above knowledge-base can have the next meaning: Let us suppose that the musicians (mu) generally (on level 0.7) love (lo) the good composers (gc). Mary (M) fairly (on 0.8 level) is a music founder (mf), and Vivaldi (V) is generally (on 0.9 level) is a favourite composer (fv). It is known that Vivaldi and Bach (B) are similar composers. How much does Mary like (li) Bach?
Our experience is: according to the algorithm mA1 we can't reply to this question, but there is an answer in the case of algorithm mA2.

6 Evaluation strategies

The consequence of a fuzzy knowledge-base is defined as fixpoint semantics. This means that starting from the facts, applying the rules and proximities we can deduce all reasonable atoms. This kind of evaluation is called "bottom up" deduction. However many times there is no need for full evaluation, because we want to answer a concrete question, we want to know the truth value of a statement or we want to determine its uncertainty level. This means, that in the knowledge of the goal, it is enough to make the evaluation "expediently", that is it is enough to take into consideration only those rules which are needed to reach the goal. This kind of evaluating strategy, which starts from the goal and deduces in the direction of facts is called "top down evaluation". To apply this strategy we have to complete the knowledge-base with a goal (query), that is with a pair ($q(t_1, t_2,\ldots, t_n)$, α), where $q(t_1, t_2,\ldots, t_n)$ is an atom, and α is the uncertainty level of this atom. Both the arguments and the uncertainty level can be given or wanted values. In the next section we deal with top-down evaluation of knowledge-base.

6.1 Evaluating of knowledge-base based on algorithm mA1

The top down evaluation of fuzzy Datalog is treated in [A1] and [AK3], its extension for fuzzy knowledge-base is discussed in [A2] and [A3]. Now we are going to give a short summary about this.

Generally the top down evaluation works through sub-queries. This means, that all possible rules are selected whose head can be unify with the given goal, and the atoms of the body are considered as new sub-goals. This procedure continues until obtaining the facts. In the case of fuzzy Datalog, the evaluation doesn't terminate obtaining the facts, because we need to determine the uncertainty level of the goal.

The evaluation is executed by the aid of an evaluation tree. This is a special hyper-graph, based on the AND/OR tree concept. The root of the tree is the goal-atom, every odd edge of this tree is an n-order hyper-edge with the set-node of n elements, and every even edge is an ordinary edge with one node. On every even level of the graph there are sub-goals, that is suitably unified rule-heads, and on every odd level there are rule-bodies. The leaves of the tree are the symbols YES or NO: if the sub-goal is unified with a fact, than there is YES, else there is NO.

The ordinary edges of this graph are labelled: this label contains the applied unification, the uncertainty level and the implication operator of the suitable rule, represented by this edge. The answer can be obtained from the labels on the ordinary edges of hyper-path ending in symbol YES: starting from the leaves, going backward to the root, the uncertainty levels can be calculated from these labels. This procedure can be extended for stratified fDATALOG too.

In the knowledge-base connected according to algorithm mA1, the modified program, mP has the same structure as the original one, so in the case of a given goal, mP can also be evaluated in top-down manner. To do this, using the proximity sets, we have to modify the original goal $(q(x_1, x_2,\ldots, x_n); \alpha)$, to the modified goal $(Q(X_1, X_2,\ldots, X_n); \alpha)$. For this query we get an answer according to the above procedure. Applying the suitable decoding functions we can decide an answer-set, which contains the required answer. The decoding algorithm can be found below:

Algorithm 2.

Procedure decoding (Q, SR, SC, Φ_P, Answers)
    S := set of answers for the query (Q; α)
    Answers := $\varnothing$
    while not(empty(S)) do
        $(Q(T_1, T_2 , \ldots, T_n); \alpha)$:= first element of S
        Answers := Answers $\cup$ decoded($(Q(T_1, T_2 , \ldots, T_n); \alpha)$)
            /* *all of the decoded answer for the goal*
             $(q(t_1, t_2,\ldots t_n); \alpha)$ *is produced* */
        S := S – $\{(Q(T_1, T_2 , \ldots, T_n); \alpha)\}$
    endwhile
endprocedure
function decoded($(Q(T_1, T_2 , \ldots, T_n); \alpha)$)
    Decoded_set := $\varnothing$
    while not(empty(Q) do
        (q , λ_q) := first element of Q
        q_set := $\varnothing$
        for each $(t_i , \lambda_{t_i}) \in T_i$ do
            q_set := q_set $\cup$ $\{(q(t_1, t_2 , \ldots, t_n); \varphi_q (\alpha, \lambda_q, \lambda_{t_1} ,\ldots, \lambda_{t_n}))$
        endfor
        Decoded_set := Decoded_set $\cup$ q_set
            /* *all of the decoded answer for the goal*
             $(Q(T_1,T_2,\ldots,T_n);\alpha)$ *is produced*/
        Q := Q – $\{(q , \lambda_q)\}$
    endwhile
    return Decoded_set
endfunction

Based on the algorithm above, it can be easily seen:

Proposition 4.
Let Answers be the set of evaluated goals by Algorithm 2. Then

$$\text{Answers} \subseteq C(P, Bk, \Phi_P, mA1).$$

6.2 Evaluating of knowledge-base based on algorithm mA2

This kind of knowledge-bases is based on more complicated transformation, and its consequence is wider than the consequence of the first kind of knowledge-base. Therefore in this case the top-down evaluation is more desirable. Recently the pure top-down evaluation is not solved yet, because to do this, we have to solve the problem of fuzzy unification and the inversion of decoding functions.

Similarly to the evaluation of ordinary fuzzy Datalog, our evaluation will have two directions, a top-down and a bottom up turn, moreover we rather rely on the bottom up evaluation, but the selection of required starting facts takes place in a top-down way. Thus the aim of the further procedure is to decide the required starting facts, from which we can reply for the query. As only these facts are searched, therefore in the top-down part of the evaluation there are no need for the uncertainty levels, so we search only among the ordinary facts and rules. To do this, we need the concept of substitution and unification which are given for example in [A1], [CGT], [P], [U], etc. But now sometimes we also need other kind of substitutions: to substitute some predicate p or term t for their proximity sets SR_p and SC_t, and to substitute some proximity sets for their members.

In the next, for the sake of simpler terminology, the concepts goal, rules and facts are ment without uncertainty levels. An AND/OR tree arise during the evaluation, this is the *searching tree*. Its root is the goal; its leaves are one of YES or NO. The parent-nodes of YES are the required starting facts. This tree is build up by alternation of proximity-based and rule-based unification.

The rule-based unification unifies the sub-goals with the head of suitable rules, and continues the evaluating by the bodies of these rules. This unification is special in the sense that during this unification a constant can be substituted with its proximity set, and the proximity sets of terms behave as ordinary constants.

The proximity-based unification unifies the predicate symbols of sub-goals by the members of its proximity set, excepting the first and last unification. The first proximity-based unification unifies the ground terms of the goal with their proximity sets, and the last one unifies the proximity sets among the parameters of resulting facts with their members.

According to the above conception, the searching graph is built up in the following way: If the goal is on depth 0, then every successor of any node on depth 3k+2 (k=0,1,…) are in AND connection, the others are in OR connection. In detail:

The successors of goal $g(t_1,t_2,\ldots t_n)$ be all possible $g'(t'_1,t'_2,\ldots t'_n)$, where $g' \in S_g$; $t'_i = t_i$ if t_i is some variable and $t'_i = S_{t_i}$ if t_i is a ground term.

If the atom $p(t_1,t_2,\ldots t_n)$ is in depth 3k (k=1,2,…), then the successor nodes be all possible $p'(t_1,t_2,\ldots t_n)$, where $p' \in S_p$.

If the atom L is in depth 3k+1 (k=1,2,…), then the successor nodes be the bodies of suitable unified rules, or the unified facts, if L is unifiable with any fact of the program, or NO, if there is not any unifiable rule or fact. That is, if the head of rule $M \leftarrow M_1,\ldots,M_n$ (n>0) is unifiable with L, then the successor of L be $M_1\theta$, ... ,$M_n\theta$,

where θ is the most general unifier of L and M. If $L = p(t_1,t_2,\ldots t_n)$ and in the program there is any fact with predicate symbol p, then the successor nodes be all possible $p(t'_1,t'_2,\ldots t'_n)$, where $t'_i \in S_{t_i}$ if $t_i = S_{t_i}$ or $t'_i = t_i\theta$, if t_i is a variable, and θ is a suitable unification.

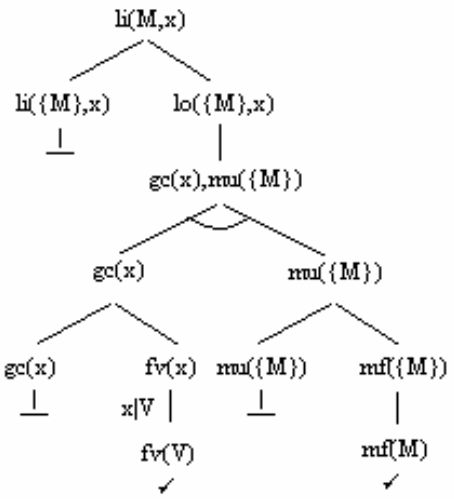
According to the previous paragraph, there are three kinds of nodes in depth 3k+2 (k=1,2,…): a unified body of a rule; a unified fact with ordinary ground term arguments; or the symbol NO. In the first case the successors are the members of the body. They are in AND connection, which is not important in our context, but maybe important for a possible future development. In the second case the successors are the symbol YES or NO, depending on whether the unified fact is among the ground atoms of the program. The NO-node has not successor.

From the construction of searching graph, it is obvious:

Proposition 5.
Let X_0 be the set of ground facts being in parent-nodes of symbols YES. Starting from X_0, the fixpoint of mNT_P contains the answer for the query.

From the viewpoint of the query, this fixpoint may contain more superfluous ground atom, but generally it is smaller than the consequence of knowledge-base. More reduction of the number of superfluous resulting facts is the work of a possible further development.

Example 7.
Let us complete the knowledge-base of example 6. by the goal li(M,x), where x is a variable. The searching graph of the query is:



Starting from the set $X_0 = \{(fv(V), 0.9), (mf(M), 0.8)\}$, now we obtain the same fixpoint as in example 6., but generally this fixpoint is tighter than the full consequence of knoledgebase.

According to the above construction, the procedure of searching the starting facts is summarized in the next algorithm:

Algorithm 3.

```
procedure evaluation(g(t), Resulting_Facts)          /* g(t) is the goal */
        Heads := {the heads of the program's rules}
        Facts := {the facts of the program}
        Resulting_Facts := ∅          /* the set of resulting starting facts */
        for all t ∈ t do
                if is_variable(t) then s := t
                                  else s := St      /* St is the proximity set of t */
                end_if
        end_for

        Nodes := {g(s)}
                                /*Nodes is the set of evaluable nodes,
                                s is the vector of elements s in the original order
*/
        New_nodes := ∅                /* the successor nodes of Nodes */
        while not_empty(Nodes) do
                p(t) := element(Nodes)
                Spnodes := ∅          /* the successor nodes of p(t) */
                proximity_evaluation(p(t),Spnodes)
                New_nodes := New_nodes ∪ Spnodes
                Nodes := Nodes – {p(t)}
        end_while
```

```
        Nodes := New_nodes
        New_nodes := ∅
        while not_empty(Nodes) do
                p(t) := element(Nodes)
                Spnodes := ∅                        /* the successor nodes of p(t) */
                rule_evaluation(p(t),Spnodes)
                New_nodes := New_nodes ∪ Spnodes
                Nodes := Nodes – {p(t)}
        end_while
end_procedure

procedure proximity_evaluation(p(t),Spnodes)
        for all q ∈ Sp do                           /* Sp is the proximity set of p */
                Spnodes := Spnodes ∪ {q(t)}
        end_for
end_procedure

procedure rule_evaluation(p(t),Spnodes)
        for all p(v) ∈ Heads do
                if  is_unifiable(p(t),p(v)) then
                  Spnodes := Spnodes ∪
                        {unified predicates of the body belonging to p(vθ)}
                                                    /* θ is the suitable unifier */
                end_if
        end_for

        for all p(v) ∈ Facts do
                if  is_unifiable(p(t),p(v)) then
                  for all St ∈ vθ do                /* θ is the suitable unifier */
                        if is_variable(St) then
                            t := Stτ                /* τ is the suitable unifier */
                        else if is_proximity_set(St) then
                            t := element(St)
                        end_if
                  end_for
                end_if
                for all possible t do
                                /* t is the vector of elements t in the right order */
                        if p(t) ∈ Facts then
                                Resulting_Facts := Resulting_Facts ∪ {p(t)}
                        end_if
                end_for
        end_for
end_procedure
```

This algorithm can be applied for stratified fDATALOG too, by determining the successor of a rule-body without negation.

7 Summary

In this paper we gave a possible model of handling uncertain information by defining fuzzy knowledge-base as a quadruple of a background knowledge which is based on the concept of proximity; a deduction mechanism, which is the fuzzy Datalog; the set of decoding functions, according to which we can compute the uncertainty level of results, and some modifying algorithm which connects the background knowledge to the deduction mechanism. We defined two kind of modifying algorithm, and gave top-down strategies to evaluate the knowledge-base. Possibly this evaluation strategy can be improved, or maybe there is a better modifying algorithm. A possible further development is to find these better solutions.

References

[A1] Ágnes Achs: Evaluation Strategies of Fuzzy Datalog Acta Cybernetica, Szeged 13 (1997) (85-102)

[A2] Ágnes Achs: Fuzzy Datalog with background knowledge Teaching Mathematics and Computer Science, Debrecen, 3 (2005) 2 (1-25)

[A3] Ágnes Achs: Fuzzy knowledge-base with fuzzy Datalog – a model for handling uncertain information, ISDA 2004 – IEEE 4th International Conference on Intelligent System Design and Application, August 26-28, 2004, Budapest (55-60)

[A4] Ágnes Achs: Computed answer based on fuzzy knowledge-base – a model for handling uncertain information, EUSFLAT-LFA 2005 – Fourth Conference of the European Society of Fuzzy Logic and Technology, September 7-9, Barcelona (94-99)

[AFG] F.Arcelli, F.Formato and G.Gerla, "Fuzzy Unification as Foundations of Fuzzy Logic programming" in Logic Programming and Soft Computing, Ed. RSP-Wiley, England, 1998.

[AK1] Ágnes Achs - Attila Kiss : Fixpoint query in fuzzy Datalog Annales Univ. Sci. Budapest, Sect. Comp. 15 (1995) (223-231)

[AK2] Ágnes Achs - Attila Kiss : Fuzzy extension of Datalog, Acta Cybernetica 12 (1995), Szeged (153-166)

[AK3] Achs Ágnes-Kiss Attila: A Datalog fuzzy kiterjesztése, Alkalmazott Matematika Lapok, Budapest, 1998. (111-138)

[CGT] S.Ceri - G.Gottlob - L.Tanca : Logic Programming and Databases, Springer-Verlag Berlin, 1990

[DP] Didier Dubois - Henri Prade: Fuzzy sets in approximate reasoning, Part 1: Inference with possibility distributions, Fuzzy Sets and Systems 40 (1991) (143-202)

[GS] Yuri Gurevich, Saharon Shelah: Fixed-point extensions of first-order logic, IEEEE Symp. on FOCS (1985), pp. 346-353.

[L] J.W.Lloyd : Foundations of Logic Programming, Springer-Verlag, Berlin, 1987.

[N] Vilém Novák : Fuzzy sets and their applications, Adam Hilger Bristol and Philadelphia, 1989.

[P] Pásztorné Varga Katalin : A matematikai logika és alkalmazásai, Tankönyvkiadó, Bp, 1986.

[S] Maria I. Sessa: Approximate reasoning by similarity-based SLD resolution, Theoretical Computer Science 275(2002) (389-426)

[SS] Michael Schroeder, Ralf Schweimeier: Arguments and Misunderstandings: Fuzzy Unification for Negotiating Agents, Electronic Notes in Theoretical Computer Science Vol. 70 (5) 2002. Elsevier Proceedings of the ICLP workshop CLIMA02, Copenhagen, Aug. 2002.

[U] J.D.Ullman : Principles of database and knowledge-base systems, Computer Science Press, Rockville, 1988.

[V1] Harry E. Virtanen: Fuzzy unification, 5th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, 1994. July 4-8, Paris

[V2] Harry E. Virtanen: Vague Domains, S-Unification and Logic Programming, Electronic Notes in Theoretical Computer Science, 66 No. 5(2002).
http://www.elsevier.nl/locate/entcs/volume66.html 18 page