3/2 (2005), 257–281
tmcs@math.klte.hu
http://tmcs.math.klte.hu

Teaching
Mathematics and
Computer Science

Fuzzy Datalog with background knowledge

Ágnes Achs

Abstract. In this paper we give a possible model for handling uncertain information. The concept of fuzzy knowledge-base will be defined as a triplet of a background knowledge defined by the similarity of predicates and terms; a deduction mechanism: a fuzzy Datalog program, and a decoding set of the program, which help us to determine the uncertainty level of the results.

Key words and phrases: fuzzy knowledge-base, fuzzy Datalog, evaluation of fuzzy knowledge-base.

ZDM Subject Classification: P05, P09, P25, P29, R45, R49, R55, R59.

1. Introduction

The study of inference systems is faced in literature with several and often very different approaches. The large part of human knowledge can't be modelled by pure inference systems, because this knowledge is often ambiguous, incomplete and vague. When knowledge is represented as a set of facts and rules, this uncertainty can be handled with the help of fuzzy logic. About the concept of deductive databases and fuzzy logic can be read in the classical works, for example in [5, 6, 7, 8, 11].

A few years ago in [3] and [1] there was given a possible combination of Datalog-like languages and fuzzy logic. In these works there was introduced the concept of fuzzy Datalog by completed the Datalog-rules and facts with an uncertainty level and an implication operator. In [4] there was given an extension of fuzzy Datalog to fuzzy relational databases.

Parallel with these works, there were researches of possible combination of Prolog language and fuzzy logic also. Several solutions were arisen for this problem, some of them are mentioned for example in [10], [12] or [2]. These solutions propose different methods for handling uncertainty. Most of them use the concept of similarity, but in various ways. More essays deal with fuzzy unification and fuzzy resolution, for example [2, 10, 12].

In this paper we give an other possible model for handling uncertain information, based on the extension of fuzzy Datalog. We build the model of fuzzy knowledge-base, which consists of a background knowledge – some similarity relations; and a fuzzy deduction mechanism – the fuzzy Datalog.

2. The fuzzy Datalog

In Datalog one can make deductions. For example, if we know, that John likes the beautiful girls, and Mary is beautiful, then we deduce that John likes Mary. But what can we do, if Mary is not so beautiful, and the beautiful girls are not so important in John's life? In fuzzy Datalog, we can complete the rules and facts with an uncertainty level. For example if Mary is only fairly beautiful, this fact can be written as a Datalog fact completed with a level:

```
beautiful('Mary'); 0.7,
```

which means, that Mary is beautiful at least 0.7 level.

The fact, that John only usually likes the beautiful girls, can be written by a rule, but this rule is completed by an implication operator and an uncertainty level:

```
likes('John',x) ← beautiful(x);I;0.8.
```

This means, that the truth-value of this fuzzy implication according to the implication operator I, is at least 0.8.

Whether how does John likes Mary? If for example, this implication operator is the Gödel-operator:

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

then the level of the rule-head can be calculated as the minimum of the level of the rule-body and the level of the rule. For John and Mary it does:

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

that is John likes Mary at least 0.7 level.

This kind of extended Datalog is called fuzzy Datalog (fDATALOG). Now we are going to summarise the concept of fDATALOG based on [3, 4]. The knowledge of basic concepts (term, atom, literal, fuzzy set, implication operator) is supposed according to for example [5, 6, 7, 8].

As it is the case for a Datalog program, also the fDATALOG program consists of rules. The notion of fuzzy rule is given in definition below:

Definition 1. A fDATALOG rule is a triplet $(r; I; \beta)$, where r is a formula of the form

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

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

In order to get 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 $(A \leftarrow ; I; \beta)$.

In the next table we give the most frequent operators:

operator	**name**	**truth value**	
$I_1(\alpha, \beta)$	Gödel	1,	if $\alpha \leq \beta$,
		β	otherwise
$I_2(\alpha, \beta)$	Lukasiewicz	1,	if $\alpha \leq \beta$,
		$1 - \alpha + \beta$	otherwise
$I_3(\alpha, \beta)$	Goguen	1,	if $\alpha \leq \beta$,
		β/α	otherwise
$I_4(\alpha, \beta)$	Kleene-Dienes	$\max(1 - \alpha, \beta)$	
$I_5(\alpha, \beta)$	Reichenbach	$1 - \alpha + \alpha\beta$	
$I_6(\alpha, \beta)$	Gaines-Rescher	1,	if $\alpha \leq \beta$,
		0	otherwise

In our case the *Herbrand universe* of a program P, denoted by H_P, is the set of all possible ground terms constructed by substituting constants occurring in P. The *Herbrand base* of P, denoted by B_P, is the set of all possible ground atoms whose predicate symbols occur in P and whose arguments are elements of H_P. A *ground instance* of a rule $(r; I; \beta)$ in P is a rule obtained from r by replacing every variable x in r by $\Phi(x)$ where Φ is a mapping from all variables occurring in r to H_P. The set of all ground instances of $(r; I; \beta)$ are denoted by $(\mathit{ground}(r); I; \beta)$. The ground instance of P is $\mathit{ground}(P) = \cup_{(r;I;\beta)\in P}(\mathit{ground}(r); I; \beta)$.

Definition 2. An interpretation of a program P, denoted by N_P, is a fuzzy set of B_P:

$$N_P \in F(B_P), \text{ that is } N_P = \bigcup_{A\in B_P} (A, \alpha_A).$$

Let α_A denote the truth value of the ground atom A.

Then $\alpha_{A_1\wedge\ldots\wedge A_n}$ and $\alpha_{\neg A}$ for ground atoms $A, A_1, \ldots, A_n$ is defined in the following way:

$$\begin{aligned} \alpha_{A_1\wedge\ldots\wedge A_n} &\overset{\text{def}}{=} \min(\alpha_{A_1}, \ldots, \alpha_{A_n}) \\ \alpha_{\neg A} &\overset{\text{def}}{=} 1 - \alpha_A. \end{aligned}$$

To be short, we sometimes denote $\alpha_{A_1\wedge\ldots\wedge A_n}$ by α_{body} and α_A by α_{head}.

Definition 3. An interpretation is a model of P if for each case of

$$(A \leftarrow A_1, \ldots, A_n; I; \beta) \in \mathit{ground}(P),$$
$$I(\alpha_{\text{body}}, \alpha_{\text{head}}) \geq \beta \text{ is true.}$$

That is for each ground rules the truth-value of the rule is at least the level of the rule.

A model M is the least model, if for any model N, we have $M \leq N$. A model M is minimal if there is no other model $N \neq M$ such that $N \leq M$.

The semantics of fDATALOG is defined as the fixpoints of consequence transformations. Depending on these transformations we defined two kind of semantics for fDATALOG. The deterministic semantics is the least fixpoint of a deterministic transformation, the nondeterministic semantics is the least fixpoint of a nondeterministic transformation. With the aid of the deterministic transformation the rules of a program are evaluated parallel, while in 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 4. The consequence transformation $NT_P\colon F(B_P) \to F(B_P)$ is defined as

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

where

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

$|A_i|$ 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).

We can define the powers of the transformations: For any $T\colon F(B_P) \to F(B_P)$ transformation let

$$\begin{aligned} T_0 = \{\cup\{(A, \alpha_A)\} \mid\ & (A \leftarrow ; I; \beta) \in \mathit{ground}(P), \\ & \alpha_A = \max(0, \min\{\gamma \mid I(1, \gamma) \ge \beta\})\} \\ \cup\, \{(A, 0) \mid \exists\, & (B \leftarrow \ldots \neg A \ldots; I; \beta) \in \mathit{ground}(P)\}, \end{aligned}$$

and let

$$\begin{aligned} T_1 &= T(T_0), \\ &\vdots \\ T_n &= T(T_{n-1}). \end{aligned}$$

In [4] it is proved, that starting from the set of facts (T_0), both deterministic and nondeterministic transformations have a fixpoint, which is the least fixpoint in the case of negation-free program. The fixpoint of nondeterministic transformation is denoted by $\text{lfp}(NT_P)$.

It was also proved, that $\text{lfp}(NT_P)$ is model of program P. This proposition is the background of the following definition:

Definition 5. We define $\text{lfp}(NT_P)$ to be the nondeterministic semantics of fDATALOG programs.

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 stratification. Stratification gives an evaluating sequence in which the negative literals are evaluated first.

To stratify a program, it is necessary to define the concept of dependency graph. This is a directed graph, whose nodes are the predicates of P. There is an arc from predicate p to predicate q if there is a rule whose body contains p or $\neg p$ and whose head predicate is q.

A program is recursive, if its dependency graph has one or more cycles.

A program is stratified if whenever there is a rule with head predicate p and a negated body literal $\neg q$, there is no path in the dependency graph from p to q.

The stratification of a program P is a partition of the predicate symbols of P into subsets $P_1, \ldots, P_n$ such that the following conditions are satisfied:

(1) if $p \in P_i$, $q \in P_j$ and there is an edge from q to p then $i \geq j$,

(2) if $p \in P_i$, $q \in P_j$ and there is a rule with the head p whose body contains $\neg q$, then $i > j$.

A stratification specifies an order of evaluation. First we evaluate the rules whose head-predicates are in P_1 then those ones whose head-predicates are in P_2 and so on. The sets $P_1, \ldots, P_n$ are called the strata of the stratification.

A program P is called stratified if and only if it admits stratification.

In [5, 11] there is a very simple method for finding a stratification for a stratified program. In [4] it is proved that for a stratified fDATALOG program P, there is an evaluation sequence, – this is the order of strata – in which $\mathrm{lfp}(NT_P)$ is minimal model of P. To be more precise, let P be a stratified fDATALOG program with stratification $P_1, \ldots, P_n$. Let P_i^* denote the set of all rules of P corresponding to stratum P_i, namely the set of all rules, whose head-predicate is in P_i.

Let

$$L_1 = \mathrm{lfp}(NT_{P_1^*}),$$

where the starting point of the computation is the set of facts.

$$L_2 = \mathrm{lfp}(NT_{P_2^*}),$$

where the starting point of the computing is L_1,

$$\ldots$$

$$L_n = \mathrm{lfp}(NT_{P_n^*}),$$

where the starting point is L_{n-1}.

In other words: at first we compute the least fixpoint L_1, corresponding to the first stratum of P. Then we can take a step to the next stratum, and so on.

It can be seen that L_n is a minimal fixpoint of P, that is $L_n = \text{lfp}(NT_P)$ ([4]).

Example 1. Given the next fDATALOG program:

$$\begin{aligned} r(a) &\leftarrow\ ; I_1; 0.8 \\ p(x) &\leftarrow r(x), \neg q(x); I_1; 0.6 \\ q(x) &\leftarrow r(x); I_1; 0.5 \\ p(x) &\leftarrow q(x); I_1; 0.8 \end{aligned}$$

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 $\text{lfp}(NT_P) = \{(r(a), 0.8); (p(a), 0.5); (q(a), 0.5)\}$.

3. Background knowledge

The facts and rules of a fDATALOG program can be regarded as any kind of knowledge, but sometime – as in the case of our model – we need other information in order to get answer for a query. For example if John likes the beautiful girls, and Mary is a pretty woman, then we know, that John maybe likes Mary, because the concept of "beautiful" and "pretty", or "girl" and "woman" are similar.

In this paragraph we give a model of background knowledge. We define similarity between predicates and between constants and these structures of similarity will serve for the background knowledge.

Definition 6. A similarity on a domain D is a fuzzy subset SD: $D \times D \rightarrow [0, 1]$ such that the following properties hold:

$$\begin{aligned} &SD(x, x) = 1 \text{ for any } x \in D && \text{(reflexivity)}, \\ &SD(x, y) = SD(y, x) \text{ for any } x, y \in D && \text{(symmetry)}. \end{aligned}$$

A similarity is transitive if

$$SD(x, z) \geq \min(SD(x, y), SD(y, z)), \text{ for any } x, y, z \in D.$$

Let us denote, that in most of practical cases the similarity is not transitive. However, there is a method for deciding transitivity, using similarity matrix for describing similarity. A similarity matrix is a matrix containing the similarity values of each pair of elements in D.

Let S be a similarity matrix. The similarity is transitive if and only if

$$S \geq S \cdot S,$$

where in the matrix-multiplication instead of multiplying elements the minimum of elements are constitute and instead of summaries the maximums are form.

In our model the background knowledge is a set of similarity sets.

Definition 7. Let $d \in D$ any element of domain D. The similarity set of d is a fuzzy subset over D:

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

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

Sometimes we are not interested in similarity when two elements are too different, namely the value of similarity is too small. Therefore we introduce the concept of the λ-cut of similarity set:

Definition 8. Let $d \in D$ any element of domain D and $0 < \lambda \leq 1$. The λ-cut of similarity set S_d is:

$$S_{d,\lambda} = \{(d_i, \lambda_i) \in S_d \mid \lambda_i \geq \lambda\}.$$

It can be shown:

Proposition 1. If the similarity is transitive, it defines λ-equivalence classifications over D.

Example 2. Let us consider the following similarity 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 similarity defined by this matrix is transitive.

Let $\lambda = 0.8$. The λ-cuts of similarity sets of $D = \{a, b, c, d, e\}$ are:

$$\begin{aligned}
S_{a,\lambda} &= \{(a,1),(c,0.8),(e,0.8)\}\\
S_{b,\lambda} &= \{(b,1),(d,0.9)\}\\
S_{c,\lambda} &= \{(a,0.8),(c,1),(e,0.8)\}\\
S_{d,\lambda} &= \{(b,0.9),(d,1)\}\\
S_{e,\lambda} &= \{(a,0.8),(c,0.8),(e,1)\}
\end{aligned}$$

These sets may define a λ-equivalence relation over D. Two elements $d_1, d_2 \in D$ are in λ-equivalence relation if $S_D(d_1, d_2) \geq \lambda$. According to this relation we can define the next equivalence classes over D:

$$E_{0.8} = \{a, c, e\}, \{b, d\}.$$

Based on similarities we can construct the background knowledge, which is an information about the similarity of terms and predicate symbols.

Definition 9. Let C be any set of ground terms and R any set of predicate symbols. Let SC and SR any similarity over C and R respectively. The background knowledge is:

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

4. Fuzzy knowledge-base

A fuzzy knowledge-base consists of a background knowledge, a fuzzy deduction mechanism, and a function, which computes the final value of the uncertainty.

4.1. Modified fDATALOG program

Let P be a fuzzy Datalog program, and let Bk be any background knowledge. With the similarities 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 be occur, that SR_p or SC_t is not in Bk, in this case $SR_p = \{(p, 1)\}$ or $SC_t = \{(t, 1)\}$.)

Algorithm 1

```
1:  procedure MODIFICATION($P$)
2:      while not(empty($P$)) do
3:          $(r; I; \beta)$ := first rule of $P$
4:          $(R; I; \beta) := ($REPLACE(r)$; I; \beta)$
5:          $P := P - \{(r; I; \beta)\}$
6:      end while
7:  end procedure

8:  procedure REPLACE($r$)
9:      $\mathrm{Pred}_r$ := set of $r$'s predicates
10:     $\mathrm{Term}_r$ := set of $r$'s ground terms
11:     $\mathrm{Var}_r$ := set of $r$'s variables
12:     while not(empty($\mathrm{Pred}_r$)) do
13:         $q$ := first predicate of $r$
14:         $Q := SP_q$
15:         $\mathrm{Pred}_r := \mathrm{Pred}_r - \{q\}$
16:     end while
17:     while not(empty($\mathrm{Term}_r$)) do
18:         $t$ := first ground term of $r$
19:         $T := ST_t$
20:         $\mathrm{Term}_r := \mathrm{Term}_r - \{t\}$
21:     end while
22:     while not(empty($\mathrm{Var}_r$)) do
23:         $x$ := first variable of $r$
24:         $X := \{x\}$
25:         $\mathrm{Var}_r := \mathrm{Var}_r - \{x\}$
26:     end while
27: end procedure
```

The modified fDATALOG program is evaluable as an ordinary fDATALOG program. There are only two problems:

(1) How can one compute the uncertainty level of the results?

(2) What can we do with the huge mass of results obtained due to the similarities?

4.2. The decoding function

The uncertainty level of the results can be obtained by using decoding functions. As the uncertainty level of the result depends on the level of the suitable result in modified program and the similarity values of the suitable predicates and the suitable terms, therefore the decoding function is a $(n+2)$-variable function defined below.

Definition 10. A decoding function is an $(n+2)$-ary function:

$$\varphi(\alpha, \lambda, \lambda_1, \ldots, \lambda_n) \colon (0,1] \times (0,1] \times (0,1] \times \cdots \times (0,1] \to [0,1]$$

so that

$$\begin{aligned} &\varphi(\alpha, \lambda, \lambda_1, \ldots, \lambda_n) \leq \min(\alpha, \lambda, \lambda_1, \ldots, \lambda_n) \text{ and} \\ &\quad \varphi(\alpha, 1, 1, \ldots, 1) = \alpha. \end{aligned}$$

Example 3.

$$\begin{aligned} \varphi_1(\alpha, \lambda, \lambda_1, \ldots, \lambda_n) &= \min(\alpha, \lambda, \lambda_1, \ldots, \lambda_n); \\ \varphi_2(\alpha, \lambda, \lambda_1, \ldots, \lambda_n) &= \min(\alpha, \lambda, (\lambda_1 \cdots \lambda_n)); \\ \varphi_3(\alpha, \lambda, \lambda_1, \ldots, \lambda_n) &= \alpha \cdot \lambda \cdot \lambda_1 \cdots \lambda_n \end{aligned}$$

are decoding functions.

We have to order decoding functions to all predicates of the program. The set of decoding functions will be the decoding set of the program:

Definition 11. Let P be a fuzzy Datalog program, and F_P be the set of the program's functors. (A functor 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). The decoding set of P is:

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

Now there are all together to define the concept of a fuzzy knowledge-base.

Definition 12. A fuzzy knowledge-base (fKB) is a triplet (Bk, P, Φ_P), where Bk is a background knowledge, P is a fuzzy Datalog program, and Φ_P is a decoding set of P.

Now with the similarities of the background knowledge we can define the modified fDATALOG program, mP. Evaluating this program we get a fuzzy set of the ground atoms of mP's Herbrand base. Applying the decoding functions to these atoms, we get the consequence of the knowledge-base. More precisely we have:

Definition 13. Let (Bk, P, Φ_P) a fuzzy knowledge-base. The consequence of the knowledge-base is given in the form

$$\begin{aligned} C(Bk, P, \Phi_P) = \{&(q(t_1, t_2, \ldots, t_n); \varphi_q(\alpha, \lambda_q, \lambda_{t_1}, \ldots, \lambda_{t_n},)) \mid \text{for each} \\ &(Q(T_1, T_2, \ldots, T_n); \alpha) \in \mathrm{lfp}(NT_{mP}), \\ &(q, \lambda_q) \in Q,\ (t_i, \lambda_{t_i}) \in T_i,\ 1 \leq i \leq n\}. \end{aligned}$$

Proposition 2.

$$\mathrm{lfp}(NT_P) \subseteq C(Bk, P, \Phi_P).$$

Proof. It is obvious according to the definitions above, because if

$$(q(t_1, t_2, \ldots, t_n); \alpha) \in \mathrm{lfp}(NT_P)$$

then $(Q(T_1, T_2, \ldots, T_n); \alpha) = (SP_q(ST_{t_1}, ST_{t_2}, \ldots, ST_{t_n}); \alpha) \in \mathrm{lfp}(NT_{mP})$, and $\varphi_q(\alpha, 1, 1, \ldots, 1) = \alpha$. □

4.3. Evaluation strategies

In [1] the author deals with the evaluation strategies of fuzzy Datalog. A fDATALOG program can be evaluated under different strategies.

The *bottom-up evaluation* starts from the facts, applies the rules, so it can deduce all computable facts, that is it can determine $\mathrm{lfp}(NT_P)$.

However in many cases, there is no need for all facts of the least fixpoint, we want to get an answer only for a concrete question. If a goal is specified together with a fDATALOG program, it is enough to consider only the rules and facts which are necessary to reach the goal. The *top-down evaluation* starts from the goal, and applies the suitable rules to reach the given facts of the program.

In the case of modified fuzzy Datalog program the bottom-up evaluation seems to be unreal in practical respect because of the huge mass of results obtained due to the similarities.

4.3.1. Top down evaluation strategy of fDATALOG programs

A *goal* is a pair $(q(t_1, t_2, \dots, t_n); \alpha)$, where $q(t_1, t_2, \dots, t_n)$ is an atom, α is the level of the atom. It is possible, that among the arguments of q there are variables or constants, and α can be either a given or a wanted value. A fuzzy Datalog extended with a goal is a *query.*

A goal is evaluated through sub-queries. This means, that all of the rules, whose head-predicate can be unificated with the given goal-predicate are selected, and the predicates of the body are considered as new sub-goals. This procedure continues until obtaining the facts. This kind of evaluation is the top-down evaluation.

To apply this strategy, we need further ideas.

Definition 14. A substitution θ is a finite set of the form $\{x_1 \,|\, t_1, \dots, x_n \,|\, t_n\}$, where x_i $(i = 1, \dots, n)$ are distinct variables and $t_i \neq x_i$ $(i = 1, \dots, n)$ are terms.

The set of variables $\{x_1, \dots, x_n\}$ is called the domain of θ. If all terms $t_1, \dots, t_n$ are constants, then θ is called a ground substitution. The empty substitution is denoted by ε.

If θ is a substitution and t is a term, then $t\theta$ denotes the term which is defined as follows (it works, because we consider only the function-free case):

$$t\theta = \begin{cases} t_i, & \text{if } t \mid t_i \in \theta, \\ t & \text{otherwise.} \end{cases}$$

If L is a literal, then $L\theta$ denotes the literal which is obtained from L by simultaneously replacing each variable x_i that occurs in L by the corresponding term t_i, iff $x_i \mid t_i$ is an element of θ.

For example, let $L = \neg p(a, x, y, b)$ and $\theta = \{x \mid c, y \mid x\}$, then

$$L\theta = \neg p(a, c, x, b).$$

If $(r; I; \beta)$ is a fDATALOG rule, then $(r\theta; I; \beta)$ denotes the rule, which is obtained simultaneously applying the substitution θ for all literals of r. The substitution can result the same atoms in the body of $r\theta$, but they are considered with single multiplicity.

Definition 15. Let $\theta = \{x_1 \mid t_1, \dots, x_n \mid t_n\}$ and $\sigma = \{y_1 \mid u_1, \dots, y_n \mid u_n\}$ be two substitutions. The composition $\theta\sigma$ of θ and σ if it exists is obtained from the set

$$\{x_1 \mid t_1\sigma, \ldots, x_n \mid t_n\sigma, y_1 \mid u_1, \ldots, y_m \mid u_m\}$$

by eliminating each component of the form $z \mid z$ and by eliminating each component for which $y_i = x_j$ for some j.

The partial composition of substitutions θ and σ is $\theta\sigma$, if both of them are defined and is not defined if any of substitutions is not defined.

If $(r; I; \beta)$ is a rule, then applying $\theta\sigma$ to the rule has the same effect as first applying θ to r, yielding $(r\theta; I; \beta)$, and then applying σ to $r\theta$.

For example, let $L = \neg p(a, x, y, b)$ and $\theta = \{x \mid c, y \mid x\}$, $\sigma = \{x \mid d\}$ then

$$L\theta\sigma = \neg p(a, c, d, b).$$

Definition 16. If for a pair of literals L and M a substitution θ exists, such that $L\theta = M\theta$, then we say that L and M are unifiable and the substitution θ is called a unifier.

Let θ and λ be substitutions. θ is more general than λ iff there exists a substitution σ such that $\theta\sigma = \lambda$.

Let L and M be two literals. The most general unifier of L and M (mgu(L,M)) is a unifier which is more general than any other unifier.

The concept of mgu has been introduced in much more general contexts, where terms may contain function symbols. There are different algorithms for determining mgu ([9, 11]). As now we deal with function-free fDATALOG, therefore it is practical to give a simple algorithm, which generates a mgu for each pair of literals L and M if they are unifiable, or tells if they are not. Algorithm 2 shows this simpler method.

Let $L = p(t_1, \ldots, t_n)$ and $M = p'(t'_1, \ldots, t'_m)$ be two literals. The function mgu(L, M) can be generated in the following way:

Algorithm 2

```
1: function MGU($L, M$)
2:     if $p \neq p'$ or $n \neq m$ then
3:         $L$ and $M$ are not unifiable
4:     else
5:         $\theta := \varepsilon$
6:         $k := 1$
```

Algorithm 2 continued

```
7:          unifiable := true
8:          while $k \leq n$ and unifiable do
9:              if $t_i\theta \neq t'_i\theta$ then
10:                 if $t'_i\theta$ is a variable then
11:                     $\theta = \theta\{t'_i\theta \mid t_i\theta\}$
12:                 else if $t_i\theta$ is a variable then
13:                     $\theta = \theta\{t_i\theta \mid t'_i\theta\}$
14:                 else
15:                     unifiable := false
16:                 end if
17:             end if
18:             $k := k + 1$
19:         end while
20:         if unifiable then
21:             MGU$(L, M) := \theta$
22:         else
23:             $L$ and $M$ are not unifiable
24:         end if
25:     end if
26: end function
```

From the algorithm it can be seen, that mgu$(L, M) \neq$ mgu(M, L). Because of this asymmetry we have to be very careful during the top-down evaluation.

For working out the top-down evaluation process, it may occur, that only some of the variables have to be substituted. For that purpose we introduce the notion of the projection of a substitution onto a set of variables.

Definition 17. Let $\theta = \{x_1 \mid t_1, \ldots, x_n \mid t_n\}$ be a substitution and let $H = \{x_{i_1}, \ldots, x_{i_k}\}$ be a set. The projection of θ to H is the substitution $\theta_H = \{x_{i_1} \mid t_{i_1}, \ldots, x_{i_k} \mid t_{i_k}\}$.

Since rules can be joint together through their predicates in common, form one's body to another head, also substitution must be applied sometimes through common variables.

Definition 18. Let $\theta = \{x_1 \mid t_1, \ldots, x_n \mid t_n\}$ and $\sigma = \{y_1 \mid u_1, \ldots, y_n \mid u_n\}$ be substitutions. Let us suppose that for each pair $x_i \mid t_i$, $y_j \mid u_j$ for which $x_i = y_j$, also $t_i = u_j$. Then the join of θ and σ is the set $\theta \otimes \sigma = \{x_1 \mid t_1, \ldots, x_n \mid t_n, y_1 \mid u_1, \ldots, y_m \mid u_m\}$, from which the repeated components are omitted. If for any pair $x_i \mid t_i$, $y_j \mid u_j$, $x_i = y_j$, but $t_i \neq u_j$, then the join of θ and σ is not defined.

First we deal with the evaluation of negation-free fDATALOG programs. For that purpose an evaluation tree is constructed. This is a special hyper-graph, based on the AND/OR tree concept. Every odd edge of the evaluation 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.

The root is the goal-atom, the leaves are the symbols ☺ and ☹, and the nodes are defined recursively, as follows.

Let the level of the root be 0. On every even level of the graph there are sub-goals, on every odd level there are bodies of rules.

Let the atom L be a node of level $k = 2i$, and let us suppose, that there are m rules in the form

$$M \leftarrow M_1, \ldots, M_n; I; \beta,$$

whose heads are unifiable with L. Then this node has m children, and these children are in the form

$$M_1\theta, \ldots, M_n\theta,$$

where $\theta = \mathrm{mgu}(L, M)$, if $n > 0$; if $n = 0$, then the child is the symbol ☺.

If there are not any unifiable rule, then the child is the symbol ☹.

We have to pay special attention to rename the variables, since the variables in the body of a unified rule must be different from the former unifications. To solve this problem, we will identify these variables by subscribing them with the level of the evaluation tree.

Let us attach labels to the even edges of the graph. The label of edge $L \to M_1\theta, \ldots, M_n\theta$ is a triplet $(\theta; I; \beta)$.

The odd hyper-edges do not have labels.

An answer for the query can be got from the labels of the evaluating tree.

The path ending in the symbol ☹ doesn't give solution. If there is a path from one node of a hyper-edge to the symbol ☹, all of the nodes belonging to this hyper-edge and their descendants are cancelled. The resulting graph is called searching graph.

A solution can be achieved along the path ending in the symbol ☺ in the searching graph. The union of these solutions is the answer to the given query. The levels of the atoms in the answer can be computed by the uncertainty-level function, defined next.

Definition 19. The function

$$f(I,\alpha,\beta) = \min(\{\gamma \mid I(\alpha,\gamma) \geq \beta\})$$

is called uncertainty-level function.

In the case of the studied implication operators $f(I,\alpha,\beta)$ is the following:

$$\begin{aligned}
f(I_1,\alpha,\beta) &= \min(\alpha,\beta),\\
f(I_2,\alpha,\beta) &= \max(0,\alpha+\beta-1),\\
f(I_3,\alpha,\beta) &= \alpha\cdot\beta,\\
f(I_4,\alpha,\beta) &= \begin{cases} 0, & \text{if } \alpha+\beta \leq 1,\\ \beta, & \text{if } \alpha+\beta > 1,\end{cases}\\
f(I_5,\alpha,\beta) &= \max(0,1+(\beta-1)/\alpha),\ \alpha \neq 0,\\
f(I_6,\alpha,\beta) &= \alpha.
\end{aligned}$$

The substitution θ along the hyper-path leading to the symbol ☺ can be determined in the following way (containing hyper-edges, a path may end in more leaves):

For each hyper-node the join of the substitutions of the body's atoms is taken. Ordering this joins to the nodes of even levels (that is, to the nodes of the heads), the partial composition of these substitutions gives the substitution along the hyper-path.

One answer to the query (L,α) is $(L\theta,\alpha_{\text{goal}})$, where α_{goal} can be computed recursively by the uncertainty-level function $f(I,\alpha,\beta)$ in the following way:

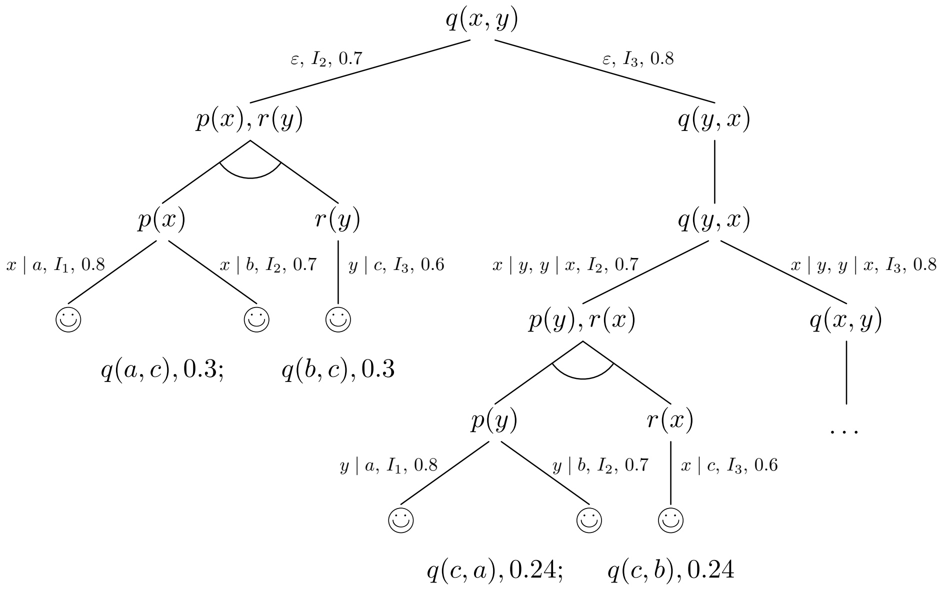
Starting at the leaves, we order to them the value $\alpha = 1$. In such away we go backward to the root. If the uncertainty level of a node on the odd level of the graph is α, then the uncertainty level of the parent node is $\alpha = f(I,\alpha,\beta)$, where $I;\beta$ are the values in the label of the edge. If the uncertainty level of the children of a node on the odd level of the graph is $\alpha_1,\ldots,\alpha_k$, then let the uncertainty level of the node be $\alpha = \min(\alpha_1,\ldots,\alpha_k)$. Finally we order an uncertainty level to the root. This level is the α_{goal}.

The uncertainty level of the goal $(q(t_1,t_2,\ldots,t_n);\alpha)$ is either constant or a variable. If it is a variable, this variable gets value during the evaluation. If α is a constant, then the uncertainty level received during the execution of the algorithm is a solution only in that case, if this level is greater then α. In this case, however, it is unnecessary to consider all the rules of the program. It is enough to take the rules, whose uncertainty factors are greater then α. Thus, the size of the evaluation graph can be reduced.

Example 4. Suppose we want to get an answer for query $(q(x,y);\alpha)$ from the fDATALOG program below:

$$\begin{aligned} p(a) &\leftarrow\ ; I_1; 0.8\\ p(b) &\leftarrow\ ; I_2; 0.7\\ r(c) &\leftarrow\ ; I_3; 0.6\\ q(x,y) &\leftarrow p(x), r(y); I_2; 0.7\\ q(x,y) &\leftarrow q(y,x); I_3; 0.8\\ s(x) &\leftarrow q(x,y); I_3; 0.9 \end{aligned}$$

The searching tree for this query:



According to the above AND/OR graph, the solution is:

$$\{(q(a,c),0.3),\ (q(b,c),0.3),\ (q(c,a),0.24),\ (q(c,b),0.24)\}$$

It can be seen, that in the case of finite evaluation graph the bottom-up and the top-down strategy give the same result. More exactly, as in [1] is proven:

THEOREM 1. *For a given goal and in the case of finite evaluation graph, the top-down evaluation gives the same result as the fixpont query.*

In [1] there is an algorithm to evaluate this graph. The algorithm consists of two procedures calling each other, one of these procedures evaluating a goal or a sub-goal, the other evaluating a rule-body.

The GOAL_EVALUATION procedure determines all of the unified bodies in the case of unificable rules, and evaluating these bodies gives the answer to the goal. The RULE_EVALUATION procedure evaluating the sub-goals of the body gives the substitution belonging to the body and the uncertainty level of the body.

The order of the unificable rules in the GOAL_EVALUATION and that of the sub-goals in the RULE_EVALUATION are determined by a selection function (RULE_SELECTION and ATOM_SELECTION respectively). This order doesn't have an effect on the result, but it has influence on the efficiency of evaluation. The special symbols, ☺ and ☹ are not in the set of evaluable sub-goals, because they are not evaluable. (In the case of ☹ there is no unificable rule, in the case of ☺ we get an empty node after unifying, so we can determine the answer immediately.)

It is practical to solve the join of the substitutions in a top-down manner, that is, not to consider the sub-goals as independent evaluations, but to narrow the size of the graph by a "sideways information passing". This means, that the substitution getting by evaluation of a sub-goal, can be applied immediately to the other members of the body, so the number of examinable paths can be reduced.

During the evaluation of a sub-goal, it is enough to consider only the projection of the substitution to the variables of the sub-goal.

If it is necessary, the variables can be renamed using the set of substituting-terms. The set of substituting-terms of substitution $\theta = \{x_1 \mid t_1, \dots, x_n \mid t_n\}$ is the set $\{t_1, \dots, t_n\}$.

This algorithm provides the answer in the case of a given program and a given goal.

Algorithm 3

```
1: EVALUATION:
2: begin
3:     solution := $\emptyset$
4:     goalanswer := $\emptyset$
```

Algorithm 3 continued

```
5:     GOAL_EVALUATION(goal, goalanswer)        ▷ goalanswer is the set of
                 ▷ all possible result pair of substitution and uncertainty level
6:     while not_empty(goalanswer) do
7:         ($\theta, \alpha$goal) := first element (goalanswer)
8:         goalanswer := goalanswer $-$ {($\theta, \alpha$goal)}
9:         solution := solution $\cup$ {(goal's_atom$\theta, \alpha$goal)}
                 ▷ the result substitutions are applied for the atom of the goal
10:    end while
11: end

12: procedure GOAL_EVALUATION(goal, goalanswer)
13:    goal_variables := {the set of the variables of the goal}
14:    $R$ := {($r; I; \beta$) | rule's_head($r$) is unificable with the goal}
15:    if $R = \emptyset$ then return
16:    end if
17:    while not_empty($R$) do
18:        ($r; I; \beta$) := rule_selection($R$)
19:        $R := R - \{(r; I; \beta)\}$
20:        body := rule's_body($r$)
21:        for all variable $\in r$ do
22:            if variable $\in$ substituting_terms($\theta$) then
23:                variable := newname(variable)
24:            end if
25:        end for
26:        $\theta$ := MGU(goal's_atom, rule's_head($r$))
27:        body := body$\theta$
28:        $\alpha$body := 1
29:        $\theta$body := $\varepsilon$
30:        if body $= \emptyset$ then
31:            goalanswer := goalanswer $\cup$ {($\theta, f(I, \alpha$body$, \beta)$)}
32:        else
33:            RULE_EVALUATION(body, $\alpha$body, $\theta$body, goalanswer, goal_variables, $I$, $\beta$)
                                             ▷ the body of the rule is evaluated
34:        end if
35:    end while
36: end procedure
```

Algorithm 3 continued

```
37: procedure RULE_EVALUATION(body, $\alpha$body, $\theta$body, goalanswer, goal_variables, $I$, $\beta$)
38:     atom := atom_selection(body)
39:     newbody := body $-$ {atom}
40:     answer := $\emptyset$
41:     GOAL_EVALUATION(atom, answer)
42:     if answer = $\emptyset$ then return
43:     end if
44:     while not_empty(answer) do              ▷ the remaining part of the body
                                                ▷ is evaluated for all possible substitution
45:         $(\theta, \alpha\text{atom})$ := element(answer)
46:         answer := answer $- \{(\theta, \alpha\text{atom})\}$
47:         $\theta$body := $\theta$body$\theta$
48:         $\alpha$body := min($\alpha$body, $\alpha$atom)
49:         if newbody $\neq \emptyset$ then
50:             newbody := newbody$\theta$
51:             RULE_EVALUATION(newbody, $\alpha$body, $\theta$body, goalanswer, goal_variables, $I$, $\beta$)
52:         end if
53:         if newbody = $\emptyset$ then
54:             $\theta$ := projection($\theta$body, goal_variables)
55:             goalanswer := goalanswer $\cup \{(\theta, f(I, \alpha\text{body}, \beta))\}$
56:         end if
57:     end while
58: end procedure
```

In the case of recursive programs, the top-down evaluation may not terminate (as in the case of Example 4). But if we order a depth limit to each recursive atom, the procedure can be stopped. [1] determines a suitable depth limit. Of course, a bad limit can cause result-loss.

It is easy to apply the top-down evaluation for stratified fDATALOG. In the case of stratified fDATALOG, the head-predicate of a rule is at least as high stratum as the predicates of the body. In other words, during the top-down evaluation we approach from the higher strata to the lower ones, that is, in the evaluation graph the stratum of a parent node is not lower than the stratum of the children. Therefore, when we compute the uncertainty level, we are starting at the lowest stratum. This observation can be used to handle the negated predicates. If a sub-goal is negated, let us indicate this sub-goal, and pay attention to this marking during the computation of the uncertainty level. If the atom is marked and the uncertainty level α is computed up to this point, then the computation continues with value $1-\alpha$.

4.3.2. Top down evaluation strategy of fuzzy knowledge-base

A query over the fuzzy knowledge-base (Bk, P, Φ_P) can be defined also.

According to Algorithm 1 program P can be turned into mP. Similarly we can modify the goal $(q(x_1, x_2, \ldots, x_n); \alpha)$ into $(Q(X_1, X_2, \ldots, X_n); \alpha)$.

The modified fDATALOG program is evaluable as an ordinary fDATALOG program, so we get answer for the modified goal. Applying the decoding function the answers for the query over the fuzzy knowledge-base can be obtained.

The next algorithm gives the set of answers to the goal $(q(t_1, t_2, \ldots, t_n); \alpha)$ by decoding the answers for the query $(Q(X_1, X_2, \ldots, X_n); \alpha)$:

Algorithm 4

```
 1: procedure DECODING($Q, SP, ST, \Phi_P$)
 2:     $S$ := set of answers for the query $(Q; \alpha)$
 3:     Answers := $\emptyset$
 4:     while not(empty($S$)) do
 5:         $(Q(T_1, T_2, \ldots, T_n); \alpha)$ := first element of $S$
 6:         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 produce
 7:         $S := S - \{(Q(T_1, T_2, \ldots, T_n); \alpha)\}$
 8:     end while
 9: end procedure

10: function DECODED($(Q(T_1, T_2, \ldots, T_n); \alpha)$)
11:     Decoded_set := $\emptyset$
12:     while not(empty($Q$)) do
13:         $(q, \lambda_q)$ := first element of $Q$
14:         q_set := $\emptyset$
15:         for each $(t_i, \lambda_{t_i}) \in T_i$ do
16:             q_set := q_set $\cup \{(q(t_1, t_2, \ldots, t_n); \varphi_q(\alpha, \lambda_q, \lambda_{t_1}, \ldots, \lambda_{t_n}))\}$
17:         end for
18:         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 produce
19:         $Q := Q - \{(q, \lambda_q)\}$
```

Algorithm 4 continued

```
20:     end while
21:     return Decoded_set
22: end function
```

Algorithm 4 ensures the following statement:

PROPOSITION 3. Let Answers be the set of evaluated goals by Algorithm 4. Then

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

For better understanding Example 5 illustrates the concepts discussed above.

EXAMPLE 5. Let us see the program of Example 4:

$$\begin{array}{rcl} p(a) & \leftarrow & ; I_1; 0.8 \\ p(b) & \leftarrow & ; I_2; 0.7 \\ r(c) & \leftarrow & ; I_3; 0.6 \\ q(x,y) & \leftarrow & p(x), r(y); I_2; 0.7 \\ q(x,y) & \leftarrow & q(y,x); I_3; 0.8 \\ s(x) & \leftarrow & q(x,y); I_3; 0.9 \end{array}$$

The background knowledge is given as follows:

$SP =$

	p	q	r	s
p	1	0.8		
q	0.8	1	0.7	
r		0.7	1	
s				1

$ST =$

	a	b	c
a	1		0.9
b		1	0.7
c	0.9		1

The decoding functions are defined as:

$$\begin{array}{rcl} \varphi_p(\alpha, \lambda, \lambda_1) & = & \min(\alpha, \lambda, \lambda_1); \\ \varphi_q(\alpha, \lambda, \lambda_1, \lambda_2) & = & \alpha \cdot \lambda \cdot \min(\lambda_1, \lambda_2); \\ \varphi_r(\alpha, \lambda, \lambda_1) & = & \min(\alpha, \lambda, \lambda_1); \\ \varphi_s(\alpha, \lambda, \lambda_1) & = & \alpha, \lambda, \lambda_1. \end{array}$$

Let the goal be the same as it is in Example 4: $(q(x,y); \alpha)$.

The necessary part of modified fDATALOG for $(Q(X,Y);\alpha)$ goal is:

$$\begin{aligned} P(A) &\leftarrow\ ; I_1; 0.8 \\ P(B) &\leftarrow\ ; I_2; 0.7 \\ R(C) &\leftarrow\ ; I_3; 0.6 \\ Q(X,Y) &\leftarrow\ P(X), R(Y); I_2; 0.7 \\ Q(X,Y) &\leftarrow\ Q(Y,X); I_3; 0.8 \end{aligned}$$

where

$$\begin{aligned} P &= \{(p,1),(q,0.8)\}; & Q &= \{(q,1),(r,0.7),(p,0.8)\}; & R &= \{(r,1),(q,0.7)\}; \\ A &= \{(a,1),(c,0.9)\}; & B &= \{(b,1)\}; & C &= \{(c,1),(a,0.9)\}. \end{aligned}$$

According to Example 4, the answer for this query is:

$$\{(Q(A,C),0.3),\ (Q(B,C),0.3),\ (Q(C,A),0.24),\ (Q(C,B),0.24)\}$$

Applying the decoding algorithm, these modified atoms can be decoded: $\{(q(a,c),0.3)$, $(q(a,a),0.27)$, $(q(c,a),0.243)$, $(q(c,c),0.27)$, $(r(a,c),0.21)$, $(r(a,a),$ $0.189)$, … etc.$\}$; $\{(q(b,c),0.3)$, … etc.$\}$; etc.

This example shows, that even a case of simple knowledge-base and a simple query, we can obtain a huge mass of resulting facts. To reduce the number of these, we can extend the query with the levels of desirable similarities, so the wanted goal is:

$$(q(t_1,t_2,\dots,t_n);\alpha;\lambda_R;\lambda_C),$$

where $q(t_1,t_2,\dots,t_n)$ is the goal-atom, α is the uncertainty level of the goal (constant or variable), λ_R, λ_C are the acceptable similarity degree of predicates and terms respectively.

Evaluating this goal – in case of given λ_R, λ_C – in the modified fDATALOG the similarity sets are replaced by their λ_R or λ_C-cuts.

Note: In case of transitive similarities the number of rules and facts to be evaluated is reduced significantly and therefore so does the number of resulting facts.

5. Summary

In this paper a possible model of handling uncertain information was given by defining fuzzy knowledge-base as a triplet of a background knowledge, a deduction mechanism and a decoding set, and a possible evaluation strategy was shown.

By this model one can not handle all kind of uncertainty. Therefore developments of further models are desirable.

References

[1] Ágnes Achs, Evaluation Strategies of Fuzzy Datalog, *Acta Cybernetica* **13**, Szeged (1997), 85–102.

[2] F. Arcelli, F. Formato and G. Gerla, Fuzzy Unification as Foundations of Fuzzy Logic programming, in: *Logic Programming and Soft Computing*, RSP–Wiley, England, 1998.

[3] Ágnes Achs and Attila Kiss, Fixpoint query in fuzzy Datalog, *Annales Univ. Sci. Budapest, Sect. Comp.* **15** (1995), 223–231.

[4] Ágnes Achs and Attila Kiss, Fuzzy extension of Datalog, *Acta Cybernetica* **12**, Szeged (1995), 153–166.

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

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

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

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

[9] Pásztorné Varga Katalin, *A matematikai logika és alkalmazásai*, Tankönyvkiadó, Budapest, 1986.

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

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

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

ÁGNES ACHS
UNIVERSITY OF PÉCS, FACULTY OF ENGINEERING, DEPARTMENT OF COMPUTER SCIENCE
H–7624 PÉCS, BOSZORKÁNY U. 2
HUNGARY

E-mail: achs@witch.pmmf.hu

(Received June, 2005)