Acta Cybernetica 12 (1995) 153-166.

Fuzzy Extension of Datalog*

Ágnes Achs † Attila Kiss‡

Abstract

In this paper we define the fuzzy Datalog programs as sets of Horn-formulae with degrees and give their meaning by defining the deterministic and nondeterministic semantics. In the second part of the paper we show a possible extension of fDATALOG on fuzzy data.

1 Introduction

In knowledge-base systems there are given some facts representing certain knowledge and some rules which in general mean that certain kinds of information imply other kinds of information. In classical deductive database theory ([CGT], [U]) the Datalog-like data model is widely spread. Its most general type allows the use of both function symbols and negation. The meaning of a Datalog-like program is the least (if it exists) or a minimal model which contains the facts and satisfies the rules. This model is generally computed by a fixpoint algorithm.

The aim of this paper, which is partially a further development of [AK] and [K], is to give a possible extension of Datalog-like languages to fuzzy relational databases using lower bounds of degrees of uncertainty in facts and rules. We give a method for fixpoint queries. We show that this fixpoint is minimal under certain conditions.

We define the deterministic and nondeterministic semantics of fDATALOG and give a possible extension on fuzzy data.

2 The Concept of Fuzzy Datalog Program

To define the idea of fuzzy Datalog program (fDATALOG) we need some basic concepts.

A *term* is a variable, a constant or a complex term of the form $f(t_1, \ldots, t_n)$, where f is a function symbol and $t_1, \ldots, t_n$ are terms. An *atom* is a formula of the

*This research was supported by the Hungarian Foundation for Scientific Research Grant (OTKA) 2149 and OTKA 4281

†Janus Pannonius University, Pollack Mihály College, Pécs, Boszorkány u. 2, Hungary

‡Eötvös Lóránd University, Budapest, Múzeum krt. 6-8., Hungary E-mail: kiss@tomx.elte.hu

form $p(\underline{t})$, where p is a predicate symbol of a finite arity (say n) and $\underline{t}$ is a sequence of terms of length n (arguments). A *literal* is either an atom (a positive literal) or the negation of an atom (a negative literal).

A term, atom, literal is *ground* if it is free of variables.

An *implication operator* is a mapping of the form

$$I(x,y) = \begin{cases} 1 & \text{if } x \leq y \\ f(x,y) & \text{otherwise} \end{cases}$$

where $x, y \in [0,1]$ and $0 \leq f(x,y) \leq 1$.

Let D be a set. The *fuzzy set* F over D is a function $F : D \to [0,1]$. Let $\mathcal{F}(D)$ denote the set of all fuzzy sets over D. So $F \in \mathcal{F}(D)$.

$$F \cup G(d) \stackrel{\text{def}}{=} \max(F(d), G(d))$$

$$F \cap G(d) \stackrel{\text{def}}{=} \min(F(d), G(d)).$$

We can define an ordering relation: $F \leq G$ iff $F(d) \leq G(d)$, for $d \in D$.

The support of fuzzy set F is a classical set

$$\text{Supp}(F) = \{d | F(d) \neq 0\}.$$

We can see that $(\mathcal{F}(D), \leq)$ is a complete lattice. The top element of the lattice is $U : D \to [0,1] : U(d) = 1$, for $d \in D$. The bottom element is: $\emptyset : D \to [0,1] : \emptyset(d) = 0$, for $d \in D$.

Fuzzy sets are frequently denoted in the following way:

$$F = \bigcup_{d \in D} (d, \alpha_d),$$

where $(d, \alpha_d) \in D \times [0,1]$.

In general the (d, α_d) pairs where $\alpha_d = 0$ are omitted from F, and sometimes $\text{Supp}(F)$ in enlarged with $(d, 0)$ pairs, where $d \in D$ but $d \notin \text{Supp}(F)$.

Below we will define the fuzzy Datalog language which is a possible extension of Datalog, using lower bounds of degrees of uncertainty in facts and rules. In this language the rules are completed with an implication operator and with a level. We allow for each formula to use any implication operator from a given set. Thus we fix a set of implication operator. We can infer the level of a rule's head from the level of the body and the level of the rule and the implication operator of the rule.

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

$$Q \leftarrow Q_1, \ldots, Q_n \qquad (n \geq 0)$$

where Q is an atom (the head of the rule), $Q_1, \ldots, Q_n$ are literals (the body of the rule); I is an implication operator and $\beta \in (0,1]$ (the level of the rule).

An fDATALOG rule is safe if

- All variables which occur in the head also occur in the body;
- All variables occuring in a negative literal also occur in a positive literal.

An fDATALOG program is a finite set of safe fDATALOG rules. Let A be a ground atom. The rules of the form $(A \leftarrow; I; \beta)$ are called facts.

The *Herbrand universe* of a program P (denoted by H_p) is the set of all possible ground terms constructed by using constants and function symbols occuring in P. The *Herbrand base* of P (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 $(ground(r); I; \beta)$. The ground instance of P is

$$ground(P) = \cup_{(r;I;\beta)\in P}(ground(r); I; \beta).$$

Definition 2 An interpretation of a program P, denoted by N_p, is a fuzzy set of B_p:

$$N_p \in \mathcal{F}(B_p), \text{ that is } N_p = \bigcup_{A\in B_p} (A, \alpha_A).$$

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

$$\alpha_{A_1\wedge\ldots\wedge A_n} \stackrel{\text{def}}{=} \min(\alpha_{A_1}, \ldots, \alpha_{A_n}),$$

$$\alpha_{\neg A} \stackrel{\text{def}}{=} 1 - \alpha_A.$$

Definition 3 An interpretation is a model of P if for each $(ground(r); I; \beta) \in \text{ground}(P)$, $ground(r) = A \leftarrow A_1, \ldots, A_n$

$$I(\alpha_{A_1\wedge\ldots\wedge A_n}, \alpha_A) \geq \beta$$

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

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

3 The Semantics of Fuzzy DATALOG

We will define two kinds of consequence transformations. Depending on these transformations we can define two semantics for fDATALOG. In this chapter we will show that the two semantics are the same in the case of positive programs, but they are different when the program has negative literals.

Definition 4 The consequence transformations $DT_p : \mathcal{F}(B_p) \to \mathcal{F}(B_p)$ and $NT_p : \mathcal{F}(B_p) \to \mathcal{F}(B_p)$ are defined as

$$DT_p(X) = \{\cup\{(A, \alpha_A)\} | (A \leftarrow A_1, \ldots, A_n; I; \beta) \in ground(P),$$

$(|A_i|, \alpha_{A_i}) \in X$ for each $1 \leq i \leq n, \alpha_A = \max(0, \min\{\gamma | I(\alpha_{\text{body}}, \gamma) \geq \beta\})\} \cup X$

and

$NT_p(X) = \{(A, \alpha_A)\} | \exists (A \leftarrow A_1, \ldots, A_n; I; \beta) \in ground(P)$,

$(|A_i|, \alpha_{A_i}) \in X$ for each $1 \leq i \leq n, \alpha_A = \max(0, \min\{\gamma | I(\alpha_{\text{body}}, \gamma) \geq \beta\})\} \cup X$,

$|A|$ denotes $p(\underline{c})$ if either $A = p(\underline{c})$ or $A = \neg p(\underline{c})$ where p is a predicate symbol with arity k and $\underline{c}$ is a list of k ground terms.

Note: $NT_p(X)$ has at most one more element than X while $DT_p(X)$ may have many new elements.

For any $T : \mathcal{F}(B_p) \to \mathcal{F}(B_p)$ transformation let

$$T_0 = \{\cup\{(A, \alpha_A)\} | (A \leftarrow; I; \beta) \in ground(P), \alpha_A = \max(0, \min\{\gamma | I(1, \gamma) \geq \beta\})\} \cup$$

$\{(A, 0) | \exists (B \leftarrow \ldots \neg A \ldots; I; \beta) \in ground(P)\}$ and let

$$T_1 = T(T_0)$$

$$\vdots$$

$$T_n = T(T_{n-1})$$

$$\vdots$$

$T_q =$ least upper bound $\{T_\gamma | \gamma < \delta\}$ if δ is a limit ordinal.

Proposition 1 Both DT_p and NT_p have a fixpoint, i.e., there exists $X \in \mathcal{F}(B_p)$ and $Y \in \mathcal{F}(B_p) : DT_p(X) = X$ and $NT_p(Y) = Y$.
If P is positive, then $X = Y$ and this is the least fixpoint. (That is for any $Z = T(Z) : X \leq Z$.)

Proof: As [CGT] and [GS] show, if T is an inflationary transformation over a complete lattice L, then T has a fixpoint. (T is inflationary if $X \leq T(X)$ for every $X \in L$). If T is monotone ($T(X) \leq T(Y)$ if $X \leq Y$), then T has a least fixpoint (see in [L]).

Since DT_p and NT_p are inflationary and $\mathcal{F}(B_p)$ is a complete lattice, thus they have an inflationary fixpoint.

If P is positive, then $DT_p = NT_p$ and this is monotone, which proves the proposition. □

We denote the fixpoints of the transformations by $lfp(DT_p)$ and $lfp(NT_p)$.

We show, that these fixpoints are models of P, so we can define the meaning of programs by these fixpoints.

Theorem 1 $lfp(DT_p)$ and $lfp(NT_p)$ are models of P.

Proof:
For $T = DT_p$ and $T = NT_p$ in $ground(P)$ there are rules in the following forms:

a. $(A \leftarrow; I; \beta)$.
b. $(A \leftarrow A_1, \ldots, A_n; I; \beta)$; $(A, \alpha_A) \in \text{lfp}(T)$ and $(|A_i|, \alpha A_i) \in \text{lfp}(T), 1 \leq i \leq n$.
c. $(A \leftarrow A_1, \ldots, A_n; I; \beta); \exists i : (|A_i|, \alpha A_i) \notin \text{lfp}(T)$.

It was shown in [AK] that in all of these cases $I(\alpha_{\text{body}}, \alpha_A) \geq \beta$.

This model for function- and negation-free fDATALOG is the least model of P and this fixpoint can be reached in finite steps as given in [AK]. □

Now we can define the semantics of fDATALOG programs.

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

The next statement is obvious by Proposition 1.:

Proposition 2 For function- and negation-free fDATALOG, the two semantics are the same.

We can choose many kinds of implication operators, but we will use only four of those which were discussed in [AK]. They are:

$$I_1(x,y) = \begin{cases} 1 & \text{if } x \leq y \\ y & \text{otherwise} \end{cases} \qquad I_2(x,y) = \begin{cases} 1 & \text{if } x \leq y \\ 1-(x-y) & \text{otherwise} \end{cases}$$

$$I_3(x,y) = \begin{cases} 1 & \text{if } x \leq y \\ y/x & \text{otherwise} \end{cases} \qquad I_4(x,y) = \begin{cases} 1 & \text{if } x \leq y \\ 0 & \text{otherwise} \end{cases}$$

As the next example shows, if the program has any negation, the two semantics are different.

Example 1

1. $r(a) \leftarrow; I_1; 0.8$
2. $p(x) \leftarrow r(x), \neg q(x); I_1; 0.6$
3. $q(x) \leftarrow r(x); I_1; 0.5$
4. $p(x) \leftarrow q(x); I_1; 0.8$

Then $\text{lfp}(DT_p) = \{(r(a), 0.8); (p(a), 0.6; (q(a), 0.5)\}$.

In nondeterministic evaluation we can get different solutions, depending on the order of applied rules.

If the order of rules is 1., 2., 3., 4. then $(\text{lfp}NT_p) = \{(r(a), 0.8);\ (p(a), 0.6;\ (q(a), 0.5)\}$, but if the evaluating order is 1., 3., 2., 4. then the $\text{lfp}(NT_p) = \{(r(a), 0.8);\ (p(a), 0.5);\ (q(a), 0.5)\}$.

□

The set $\mathrm{lfp}(DT_p)$ is not always a minimal model as shown in the example above. So in applying the deterministic semantics it is not certain that the obtained fixpoint is minimal. In the nondeterministic case however it is minimal under certain conditions. This condition is the stratification. The stratification gives an evaluating sequence in which the negative literals are evaluated at 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:

a. if $p \in P_i$ and $q \in P_j$ and there is an edge from q to p, then $i \geq j$
b. if $p \in P_i$ and $q \in P_j$ 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 a stratification.

There is a very simple method for finding a stratification for a stratified program P in [CGT], [U].

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, that is 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 T_0 defined earlier.

$$L_2 = \mathrm{lfp}(NT_{p_2^*})$$

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 we first compute the least fixpoint L_1 corresponding to the first stratum of P. Once we computed this fixpoint we can take a step to the next strata.

Note:

$$\mathrm{lfp}(NT_{p_i^*}) = \mathrm{lfp}(DT_{p_i^*}).$$

We will show by induction that L_n is a minimal model of $\mathbf{P}$. For this purpose we need the next lemma.

Lemma 1 Let P be an fDATALOG program such that for each negated predicate in a rule body there is not any rule whose head-predicate would be the same, but this predicate can occur among the facts. Then P has a least model:

$$L = \text{lfp}(NT_p)(= \text{lfp}(DT_p)).$$

Proof:

Let p be a negated predicate in a rule body of P. As there is not any new rule for p, therefore the degree of p will never change during the computation. For such P NT_p (or DT_p) is monotone and therefore $\text{lfp}(NT_p)$ is the least model of P.

□

According to this lemma, L_1 is the least fixpoint of P_1^*. Generally L_i is the least fixpoint of P_i^*, because due to the stratification of P, all negative literals of stratum i correspond to predicates of lower strata, so there is not any rule in P_i^* whose head-predicate would be this one.

From this we get the following theorem:

Theorem 2 If P is a stratified fDATALOG program then L_n is a minimal fixpoint of P.

Theorem 3 For stratified fDATALOG program P, there is an evaluation sequence, in which $\text{lfp}(NT_p)$ is a minimal model of P.

Proof:

The above construction gives this sequence. In this sequence $L_n = \text{lfp}(NT_p)$.

□

4 Connection Between fDATALOG and Fuzzy Relations

The ordinary Datalog maybe interpreted by relations such, that to every predicate p with arity k corresponds a relation P with arity k. The rows of P are those for which the predicate p is true.

Similarly it is possible to associate relations with fDATALOG predicates. The problem is that in the literature there are different definitions of fuzzy relations. Below we will deal with two kinds of fuzzy relations.

Definition 6 A first type fuzzy relation R in $D_1, \ldots, D_n$ is characterised by an n-variate membership function:

$$\mu_R : D_1 \times \ldots \times D_n \to [0, 1].$$

In other words a first type fuzzy relation is a fuzzy subset of the Cartesian Product of $D_1, \ldots, D_n$.

We will denote this relation by:

$$(R(D_1, \ldots, D_n), \mu_R).$$

Example 2 The relation $(F(X,Y),\mu_F)$ is a first type fuzzy relation, where F denotes the friends relation:

F:	X	Y	μ_F
	John	Tom	0.1
	Jim	Bob	0.6

□

It is obvious that we can relate a first type fuzzy relation to each predicate of an fDATALOG program P so that

$$t = (a_1, \ldots, a_n) \in (R(D_1, \ldots, D_n) \text{ if and only if}$$

$$(r(a_1, \ldots, a_n), \mu_R(t)) \in \text{lfp } (DT_p) \text{ or lfp } (NT_p).$$

The first type fuzzy relations show the closeness of the connection among crisp data. However, sometimes we need to use fuzzy data. So we will define the second type fuzzy relation and will give a possible extension of fDATALOG on these relations.

Definition 7 Let $D_1, \ldots, D_n$ be n universal sets and $\mathcal{F}(D_1), \ldots, \mathcal{F}(D_n)$ be all their fuzzy sets. Then a second type fuzzy relation R is defined by an n-variate membership function:

$$\mu_R : \mathcal{F}(D_1) \times \ldots \times \mathcal{F}(D_n) \to [0,1].$$

Example 3 Let R be the following:

R:	Name	Age	Salary	μ_R
	John	31	{0.8/3000, 0.7/3500}	0.7
	Tom	middle aged	3300	0.8
	Ann	young	{0.6/2000, 0.8/2500}	0.9

R is a second type fuzzy relation. □

5 Extension of fDATALOG for Fuzzy Data

We want to extend the fDATALOG programs so that the predicates of the programs can be related to second type fuzzy relations. Therefore we allow that the constants be any fuzzy data and the variables can have any fuzzy value.

Formally an fDATALOG rule is the same as above. Evaluating an extended fDATALOG program, we would have difficulties with the unification of fuzzy data, therefore we will complete these rules with similarity predicates, so the unification will be crisp and the uncertainty is expressed by the similarity.

Definition 8 $\text{sim}_D : \mathcal{F}(D) \times \mathcal{F}(D) \to [0,1]$ is a similarity predicate if it is reflexive and symmetric, that is, if $\text{sim}_D(x,x) = 1$ and $\text{sim}_D(x,y) = \text{sim}_D(y,x)$.

A similarity matrix is a matrix corresponding to similarity predicate.

A similarity is transitive if

$$\mathrm{sim}_D(x,z) \geq \max_{y \in D}\{\min(\mathrm{sim}_D(x,y), \mathrm{sim}_D(y,z))\}.$$

Definition 9 An extended fDATALOG program is an fDATALOG program on fuzzy data completed with similarity matrices.

If we want to evaluate an extended fDATALOG program, we have to transform it to an fDATALOG program. For this purpose we will build the similarity predicates into the rules. In rewriting predicates there are the following rules:

a. p is the head predicate of a fact

$$p(a_1, \ldots, a_n) \leftarrow; I; \beta.$$

Instead of this rule, we get:

$$p(x_1, \ldots, x_n) \leftarrow p(a_1, \ldots, a_n),\ \mathrm{sim}_{D_{x_1}}(a_1, x_1), \ldots,\ \mathrm{sim}_{D_{x_n}}(a_n, x_n); I; \beta,$$

where $\mathrm{sim}_{D_{x_i}}$ denotes the similarity predicate on the domain of x_i.

b. x is a variable in the rule

$$Q \leftarrow Q_1, \ldots, Q_m; I; \beta.$$

Suppose that x is in the predicates $p, p_{i_1}, \ldots, p_{i_k}$ and there is no another occurence of x in this rule:

$$p(\ldots, x, \ldots), p_{i_1}(\ldots, x, \ldots), p_{i_k}(\ldots, x, \ldots).$$

Instead of this sequence we can write:

$$p(\ldots, x, \ldots), p_{i_1}(\ldots, x_1, \ldots),\ \mathrm{sim}_{D_x}(x_1, x), p_{i_2}(\ldots, x_2, \ldots),$$

$$\mathrm{sim}_{D_x}(x_2, x),\ \mathrm{sim}_{D_x}(x_2, x_1), \ldots,$$

$$p_{i_k}(\ldots, x_k, \ldots),\ \mathrm{sim}_{D_x}(x_k, x),\ \mathrm{sim}_{D_x}(x_k, x_1), \ldots,\ \mathrm{sim}_{D_x}(x_k, x_{k-1})$$

independently of whether p is in the head or in the body and independently of the order of the predicates (because of symmetry of sim).

Then the head of the new and of the original rule will be the same.

c. x is a repeated variable in predicate p:

$$p(\ldots, x, \ldots, x, \ldots, x, \ldots) - x \text{ occurs } k+1 \text{ times.}$$

Instead of this we get:

$$p(\ldots, x, \ldots, x_1, \ldots, x_k, \ldots)\ \mathrm{sim}_{D_x}(x_1, x),\ \mathrm{sim}_{D_x}(x_2, x), \ldots,$$

$$\mathrm{sim}_{D_x}(x_k, x),\ \mathrm{sim}_{D_x}(x_1, x_2), \ldots,\ \mathrm{sim}_{D_x}(x_1, x_k), \ldots,\ \mathrm{sim}_{D_x}(x_{k-1}, x_k).$$

In this case the head of the new rule is the same if p is in the body and it is of the form $p(\ldots, x, \ldots, x_1, \ldots, x_k, \ldots)$ if p is the head predicate.

Proposition 3 Completing the rewriting of rules we get an ordinary fDATALOG program.

This program can be evaluated by deterministic or nondeterministic semantics.

Proposition 4 If sim_{D_x} is transitive then

$$\mathrm{sim}_{D_x}(x_1, x),\ \mathrm{sim}_{D_x}(x_2, x), \ldots,\ \mathrm{sim}_{D_x}(x_k, x),\ \mathrm{sim}_{D_x}(x_2, x_1), \ldots,$$

$$\mathrm{sim}_{D_x}(x_k, x_1), \ldots,\ \mathrm{sim}_{D_k}(x_k, x_{k-1})$$

can be simplified to:

$$\mathrm{sim}_{D_x}(x_1, x),\ \mathrm{sim}_{D_x}(x_2, x), \ldots,\ \mathrm{sim}_{D_x}(x_k, x).$$

Proof: We will prove that $\mathrm{sim}_{D_x}(x_2, x_1)$ can be ignored. The proof is similar for other cases. Because of transitivity

$$\mathrm{sim}_{D_x}(x_2, x_1) \geq \min(\ \mathrm{sim}_{D_x}(x_2, x),\ \mathrm{sim}_{D_x}(x, x_1)).$$

As in case of the rule $A \leftarrow A_1, \ldots, A_n$ $\alpha_{\mathrm{body}} = \min(\alpha_{A_1}, \ldots, \alpha_{A_n})$ so it is possible to leave out $\mathrm{sim}_{D_x}(x_2, x_1)$. □

We give an algorithm of rewriting an extended fDATALOG program P.

Algorithm:

```
Procedure rewriting
        facts := { the set of facts of P}
        rules := { the set of rules of P} - facts
        C := { the set of all possible constants }
                        (that is the union of domains)
        V := { the set of variables of P}

        while not_empty (facts) do
                fact := select (facts)
                arglist := { the list of arguments of the fact }
                i := 1
                body := ∅
                for a ∈ C do
                        while a ∈ arglist do
                                change (a, x_i)
                                body := body ∧ sim_Da(x_i, a)
                                i := i + 1
                        endwhile
                endfor
                facts := facts - fact
        endwhile
```

```
while not_empty (rules) do
        rule := select (rules)
        variable_lsit := {the list of variables of the rule }
        body := { the body of the rule }
        for x ∈ V do
                if x ∈ variable_list then
                        rest_list := leave_out (x, variable_list)
                        i := 1
                        while a∈ rest_list do
                                change (x, x_i)
                                body := body ∧ sim_{D_x}(x_i, x)
                                if non_transitive (sim_{D_x}) then
                                        for j :=1 to i − 1 do
                                                body := body ∧ sim_{D_x}(x_j, x_i)
                                        endfor
                                endif
                                i := i + 1
                        endwhile
                endif
        endfor
        rules:= rules - rule
endwhile
endprocedure
```

Example 4

$$p(a) \leftarrow; I_1; 0.7$$

$$r(b) \leftarrow; I_1; 0.8$$

$$q(x) \leftarrow p(x), r(x); I_1; 0.9$$

sim	a	b	c
a	1	0.8	0.1
b	0.8	1	0.7
c	0.1	0.7	1

The rewritten rules:

$$p(a) \leftarrow; I_1; 0.7$$

$$r(b) \leftarrow; I_1; 0.8$$

$$\mathrm{sim}(a, a) \leftarrow; I_1; 1$$

$$\vdots$$

$$p(x) \leftarrow p(a),\ \mathrm{sim}(x, a); I_1; 0.7$$

$$r(x) \leftarrow r(b),\ \mathrm{sim}x, b); I_1; 0.8$$

$$q(x) \leftarrow p(x_1),\ \mathrm{sim}(x_1, x), r(x_2),\ \mathrm{sim}(x_2, x),\ \mathrm{sim}(x_2, x_1); I_1; 0.9$$

It is simpler to write the solution in matrix-form, so $\mathrm{lfp}(NT_p) = \mathrm{lfp}(DT_p)$:

p :

a	b	c
0.7	0.7	0.1

r :

a	b	c
0.8	0.8	0.7

q :

a	b	c
0.7	0.7	0.7

and the similarity matrix is the original. □

Example 5

$$p(a, b) \leftarrow; I_1; 0.9$$

$$p(c, d) \leftarrow; I_1; 0.8$$

$$q(x, y) \leftarrow p(x, y); I_1; 0.7$$

$$q(x, y) \leftarrow p(x, z), q(z, y); I_1; 0.8$$

sim	a	b	c	d	e
a	1	0	0.1	0.2	0.8
b	0	1	0.9	0.1	0
c	0.1	0.9	1	0.2	0
d	0.2	0.1	0.2	1	0.1
e	0.8	0	0	0.1	1

The rewritten rules (without facts and similarity matrix):

$$p(x, y) \leftarrow p(a, b),\ \mathrm{sim}(x, a),\ \mathrm{sim}(y, b); I_1; 0.9$$

$$p(x, y) \leftarrow p(c, d),\ \mathrm{sim}(x, c),\ \mathrm{sim}(y, b); I_1; 0.8$$

$$q(x, y) \leftarrow p(x_1, y_1),\ \mathrm{sim}(x_1, x),\ \mathrm{sim}(y_1, y); I_1; 0.7$$

$$q(x, y) \leftarrow p(x_1, z_1), q(z_2, y_1)\ \mathrm{sim}(x_1, x),\ \mathrm{sim}(y_1, y),\ \mathrm{sim}(z_1, z_2); I_1; 0.8$$

The fixpoint is:

p :	a	b	c	d	e
a	0.1	0.9	0.9	0.1	0.1
b	0.2	0.1	0.2	0.8	0.1
c	0.2	0.1	0.2	0.8	0.1
d	0.2	0.2	0.2	0.2	0.1
e	0	0.8	0.8	0.1	0

q :	a	b	c	d	e
a	0.2	0.7	0.7	0.7	0.2
b	0.2	0.2	0.2	0.7	0.2
c	0.2	0.2	0.2	0.7	0.2
d	0.2	0.2	0.2	0.2	0.2
e	0.2	0.7	0.7	0.7	0.2

and the original similarty matrix. □

Example 6

$$n(a) \leftarrow; I_1; 0.9$$

$$s(x,x) \leftarrow n(x); I_1; 0.8$$

sim	a	b	c
a	1	0.7	0.1
b	0.7	1	0
c	0.1	0	1

The rewritten rules (without facts and similarity matrix):

$$n(x) \leftarrow n(a),\ \mathrm{sim}(x,a); I_1; 0.9$$

$$s(x,x_1) \leftarrow\ \mathrm{sim}(x_1,x), n(x_2),\ \mathrm{sim}(x_2,x),\ \mathrm{sim}(x_1,x_2); I_1; 0.8$$

The fixpoint is:

n :	a	b	c
	0.9	0.7	0.1

s :	a	b	c
a	0.8	0.7	0.1
b	0.7	0.7	0
c	0.1	0	0.1

6 Conclusion

In this paper we gave a possible extension of Datalog-like languages. We defined the deterministic and nondeterministic semantics of fDATALOG and using the similarity relations we gave a possible extension on fuzzy data.

References

[AK] Ágnes Achs, Attila Kiss: Fixpoint query in fuzzy Datalog, to appear in Annales Universitatis Scientiarum Budapestinensis Sectio Computatorica, Budapest

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

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

[K] Attila Kiss: On the least models of fuzzy Datalog programs, International Conference on Informaion Processing and Management of Uncertainty in Knowledge-based system, Mallorca 465-471.

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

[LL] Deyi Li, Dongbo Liu: A Fuzzy PROLOG Database System, Researc Studies Press LTD., Taunton, Somerset, England, 1990.

[N] Vilém Novák: Fuzzy sets and their applications, Adam Hilger Bristol and Philadelphia, 1987.

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

Received June, 1995