Acta Cybernetica **13** (1997) 85-102.

Evaluation Strategies of Fuzzy Datalog

Ágnes Achs*

Abstract

A fuzzy Datalog program is a set of Horn-formulae with uncertainty degrees. The meaning of a program is the fixpoints of deterministic or nondeterministic consecutive transformations. In this paper we are going to deal with the evaluation strategies of fuzzy Datalog programs. We will determine the bottom-up and top-down strategies and show their equivalence.

1 Introduction

A logical data model consists of facts and rules. The facts represent certain knowledge from which other knowledge can be deduced by the rules. In classical deductive database theory ([CGT], [U]) the Datalog-like data model is widely spread. A Datalog program is a set of Horn-clauses, that is a set of the formulae

$$A \leftarrow B_1, \ldots, B_n$$

where $A, B_i (i = 1, \ldots, n)$ are positive literals.

The meaning of a Datalog-like program is the least (if any) or a minimal model which contains the facts and satisfies the rules. This model is generally computed by a fixpoint algorithm.

In [AK2] there was given a possible extension of Datalog-like languages to fuzzy relational databases using lower bounds of degrees of uncertainty in facts and rules. This language is called fuzzy Datalog (fDATALOG). In this language the rules are completed with an implication operator and a level. We can infer the level of a rule-head from the level of the body, the level of the rule and the implication operator of the rule. We defined the deterministic and nondeterministic semantics of fDATALOG as the fixpoints of certain transformations, gave a method for fixpoint queries, and showed that this fixpoint is minimal under certain conditions.

The aim of this paper is to give some evaluation strategies of fDATALOG programs.

First we are going to summarize the concept of fDATALOG.

*Janus Pannonius University, Pollack Mihály College, Pécs, Boszorkány u. 2, Hungary, e-mail:achs@mit.pmmfk.jpte.hu

2 Basic Concepts

A *term* is a variable, constant or 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 form $p(\underline{t})$, where p is an n-arity predicate symbol 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.

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 $F \in \mathcal{F}(D)$.

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

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

An ordering relation can be defined: $F \leq G$ iff $F(d) \leq G(d)\ \forall d \in D$. As every subset of $\mathcal{F}(D)$ has least upper bound and greatest lower bound, so $(\mathcal{F}(D), \leq)$ is a complete lattice. The top element of the lattice is $U : D \to [0,1] : U(d) = 1\ \forall d \in D$. The bottom element is: $\emptyset : D \to [0,1] : \emptyset(d) = 0\ \forall d \in D$.

Fuzzy sets are frequently denoted in the following way:

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

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

To make any deduction we need the concept of *implication operator*.

The features of implication operators are summarized in [DP]. In the next table we give the most frequent operators:

symbol	name	formula
$I_1(x,y)$	Gödel	1 if $x \leq y$ y otherwise
$I_2(x,y)$	Lukasiewicz	1 if $x \leq y$ $1 - x + y$ otherwise
$I_3(x,y)$	Goguen	1 if $x \leq y$ y/x otherwise
$I_4(x,y)$	Kleene-Dienes	$\max(1 - x, y)$
$I_5(x,y)$	Reichenbach	$1 - x + xy$
$I_6(x,y)$	Gaines-Rescher	1 if $x \leq y$ 0 otherwise

3 The Concept of *f*DATALOG

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

$$Q \leftarrow Q_0, \ldots, Q_n \quad (n \geq 0)$$

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 *f*DATALOG rule is safe if

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

An *f*DATALOG program is a finite set of safe *f*DATALOG 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 occurring 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

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

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 $(ground(r); I; \beta) \in ground(P)$, $ground(r) = A \leftarrow A_1, \ldots, A_n$

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

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 sometimes denote $\alpha_{A_1 \wedge \ldots \wedge A_n}$ by α_{body} and α_A by α_{head}.

The semantics of fDATALOG is defined as the fixpoints of consequence transformations. Depending on these transformations we can define two semantics for fDATALOG. The deterministic semantics is the least fixpoint of deterministic transformation, the nondeterminic semantics is the least fixpoint of nondeterministic transformation. With the aid of the deterministic transformation the rules of a program are evaluated parallely, while in nondeterministic case the rules are considered independently one after another.

These transformations are the following:

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

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

$$(|A_i|, \alpha_{A_i}) \in X \text{ for each } 1 \leq i \leq n,$$

$$\alpha_A = \max(0, \min\{\gamma | I(\alpha_{\text{body}}, \gamma) \geq \beta\})\} \cup X$$

and

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

where $(A \leftarrow A_1, \ldots, A_n; I; \beta) \in ground(P), (|A_i|, \alpha_{A_i}) \in X,\ 1 \leq i \leq n,$

$$\alpha_A = \max(0, \min\{\gamma | I(\alpha_{\text{body}}, \gamma) \geq \beta\}$$

$|A|$ 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.

We can define the powers of the transformations:

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

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

and let

$$T_1 = T(T_0)$$

$$\ldots$$

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

In [AK2] it was proved, that starting from the set of facts (T_0), both DT_P and NT_P have a fixpoint, which is the least fixpoint in the case of positive P. These fixpoints are denoted by $\text{lfp}(DT_P)$ and $\text{lfp}(NT_P)$.

It was also proved, that $\text{lfp}(DT_P)$ and $\text{lfp}(NT_P)$ are models of P. These propositions are the background of the following definition:

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.

For function- and negation-free fDATALOG, the two semantics are the same, but they are different if the program has any negation.

The set $\text{lfp}(DT_P)$ is not always a minimal model. In nondeterministic case, however, it is minimal 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:

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].

[AK2] proves that for stratified fDATALOG program P, there is an evaluation sequence, - this is the order of strata - in which $\text{lfp}(NT_P)$ is a minimal model of P.

More detailed:

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

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

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

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

where the starting point of the computing is L_1,

$$\ldots$$

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

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 one 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 = lfp (NT_P) ([AK2]).

4 Evaluation Strategies

An fDATALOG program can be evaluated with the aid of different strategies. Starting from the facts, applying the rules, all of the computable facts can be inferred, that is lfp (DT_P) or lfp (NT_P) can be determined. In this case, we speak about *bottom-up evaluation.*

In many cases however, the whole evaluation is not necessary, because we only want to get an answer to a concrete question. If a goal is specified together with an fDATALOG program, it is enough to consider only the rules and facts which are necessary to reach the goal. In the case of starting from the goal, and applying the suitable rules we infer to the facts, we speak about *top-down evaluation.*

5 Bottom-up Evaluation

For simplicity, we denote consequence transformation with T_P. This doesn't cause any trouble, because in the case of negation-free programs the fixpoints of the two tranformations are the same, and if the program contains any negation, we will consider only the nondeterministic transformation.

The fixpoint computation is a simple iteration with the following algorithm:

Algorithm 1

```
Procedure bottom_up
        old := T_0
        new := T_P(T_0)
        while old ≠ new do
                old := new
                new := T_P(old)
        endwhile
endprocedure
```

Note: In nondeterministic case, the halt condition means that none of the rules results in any new facts.

The disadvantage of the algorithm is the great number of superflouos evaluations. There are rules which are evaluated again and again in spite of the fact, that they don't result any new facts. Therefore, it is practical to omit these rules. Whether a rule can be omitted or not, depends on the path leading to the head predicate of the rule in the dependency graph. If this path contains any circle -

that is the rule is recursive - one can not omit the rule before obtaining the fix-point. But if the path does not contain any circle, then probably it can be omitted before terminating. A rule can be omitted, if the steps of the algorithm exceed the length of the maximal path leading to the head predicate of the rule. Using this observation a modified bottom-up evaluation strategy can be acquired.

6 Modified Bottom-up Evaluation

Let $P = \cup\{(r; I; \beta)\}$. Let $h : P \to N$ be defined in the following way:

$$h(r; I; \beta) = \begin{cases} n & \text{where } n \text{ is the length of the longest loopfree path leading to} \\ & \text{the headpredicate in dependency graph} \\ \infty & \text{if the path leading to the headpredicate contains any circle} \end{cases}$$

Let $T'_n = T_{P'_n}(T_{n-1})$ where

$$P'_n = P - \{(r; I; \eta) | h(r; I; \beta) < n\}$$

The sequence T'_n has a limit, that is:

Proposition 1 For function- and negation-free program P $\exists\ m \in N : T'_m = T'_{m+1} = \ldots = T'_\infty$

Proof: Let k be the number of predicates in P, n be the arguments' number of predicate with maximum argument's number and c be the number of constants in P. Then the proposition is true for $m = kc^n$.

□

For this m let T'_m be denoted with $T'(P)$.

Proposition 2 For negation- and function-free fDATALOG program P $lfp(T_P) = T'(P)$.

Proof:

a) From the construction of $T'(P)$, $T'(P) \subseteq lfp(T_P)$.

b) Let $(A, \alpha_A) \in lfp(T_P)$.

Then there is $(r; I; \beta) \in P$, for which $(A \leftarrow A_1, \ldots, A_n; I; \beta) \in ground(r)$. Let $h(r; I; \beta) = k$. Then $(r; I; \beta) \in P'_k$, so $(A, \alpha_A) \in T'_k \subseteq T'(P)$.

□

The algorithm of modified bottom-up evaluation is the following:

Algorithm 2

Procedure bottom-up 2
    $k := 1$
    old $:= T_0$
    new $:= T_P(T_0)$
    while old $\neq$ new do
        $k := k + 1$
        old := new
        $P := P - \{(r; I; \beta) | h(r; I; \beta) < k\}$
        new $:= T_P$ (old)
    endwhile
endprocedure

7 Modified Bottom-up Evaluation in the Case of Stratified fDATALOG

The modified bottom-up evaluation can be applied in the case of stratified fDATALOG. Then we can evaluate by strata. In details:

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-predicates are in P_i.

Let

$$L_1 = lfp(NT_{P_i^*})$$

where the starting point of the computation is L_{i-1}, and $T_{P_i^*} = NT_{P_i^*} = DT_{P_i^*}$.

Because, due to the stratification of P, all negative literals of stratum i correspond to predicates of lower strata, the evaluation of P_i^* is the same as the evaluation of a negation-free program.

From this the following proposition can be made:

Proposition 3 L_i can be evaluated by the modified bottom-up evaluation, that is $L_i = T'(P_i^*)$.

8 Top-down Evaluation

In many cases we only want to get an answer to a concrete question. In such cases a goal is specified together with an fDATALOG program. Then during the evaluation it is enough to consider only the rules and facts which are necessary to reach the goal.

A *goal* is a pair $(Q\alpha)$, where Q is an atom, α is the level of the atom. It is possible, that Q contains variables, and α can be either a constant or a variable. An fDATALOG program enlarged with a goal is a *query*.

A goal can be evaluated with the aid of 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 deal with this strategy, we need some basic concepts.

Definition 6 A substitution θ is a finite set of the form $\{x_1|t_1,\ldots,x_n|t_n\}$, where $x_i(i=1,\ldots,n)$ is a distinct variable and $t_i \neq x_i$ $(i=1,\ldots,n)$ is a term. The set of variable $\{x_1,\ldots,x_n\}$ is called the domain of θ. If all terms $t_1,\ldots,t_n$ 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:

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

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 $x_i|t_i$ is an element of θ.

For example, let $L = \neg p(a,x,y,b)$ and $\theta = \{x|c,y|x\}$, 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. In the body of $r\theta$ the atoms are considered with single multiplicity.

Definition 7 Let $\theta = \{x_1|t_1,\ldots,x_n|t_n\}$ and $\sigma = \{y_1|u_1,\ldots,y_n|u_n\}$ be two substitutions. The composition $\theta\sigma$ of θ and σ is obtained from the set

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

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

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 $r\theta$.

Definition 8 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. We say that θ is more general than λ iff a substitution σ such that $\theta\sigma = \lambda$ exists.

Let L and M be two literals. A 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* ([P], [U]). 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.

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 3

Function $mgu(L, M)$
    if $p \neq p'$ or $n \neq m$ then L and M are not unifiable
    else
        $\theta := \varepsilon$
        $k := 1$
        unifiable := true
        while $k \leq n$ and unifiable do
            if $t_i\theta \neq t'_i\theta$
                then if $t'_i\theta$ is a variable
                    then $\theta := \theta\{t'_i\theta | t_i\theta\}$
                    else if $t_i\theta$ is a variable
                        then $\theta := \theta\{t_i\theta | t'_i\theta\}$
                        else unifiable := false endif
                    endif
            endif
            $k := k + 1$
        endwhile
        if unifiable then $mgu(L, M) = \theta$ else L and M are not unifiable
    endif
endfunction

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

We also need the concept of projection and join of substitutions.

Definition 9 Let $\theta = \{x_1|t_1, \ldots, x_n|t_n\}$ be substitution and let $H = \{x_{i_1}, \ldots, x_{I_k}\}$ be a set. The projection of θ to H is the substitution $\theta_H = \{x_{i_1}|t_{i_1}, \ldots, x_{i_k}|t_{i_k}\}$.

Definition 10 Let $\theta = \{x_1|t_1, \ldots, x_n|t_n\}$ and $\sigma = \{y_1|u_1, \ldots, y_n|u_n\}$ be substitutions. Let us suppose that for each pair $x_i|t_i, y_j|u_j$ for which $x_i = y_j$ is true, $t_i = u_j$ also comes true. Then the join of θ and σ is the set $\theta \otimes \sigma = \{x_1|t_1, \ldots, x_n|t_n, y_1|u_1, \ldots, y_m|u_m\}$, from which the repeated components are omitted.

If for any pair $x_i|t_i, y_j|u_j, x_i = y_j$ is true, but $t_i \neq u_j$, then the join of θ and σ is not defined.

From this definition one can see, that the join is a partial operation. If we want to apply the join and the composition together, the concept of partial composition has to be defined.

Definition 11 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.

First we deal with the evaluation of negtion-free fDATALOG programs. We will search the solution with the aid of evaluation graph. This is a special AND/OR

tree, a special hyper-graph. Every odd edge is a n-order hyper-edge with the set-node of n elements, and every even edge is an ordinary edge with one node. More precisely:

An evaluation hypergraph is a tree, whose root is the goal, the leaves are the symbols "good" and "bad", and the nodes are defined recursively.

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

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

$$R \leftarrow R_1, \ldots, R_n; I; \beta$$

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

$$R_1\theta, \ldots, R_n\theta$$

where $\theta = mgu(Q, R)$,if $n > 0$; if $n = 0$, then the child is the symbol "good". If there are not any unifiable rule, then the child is the symbol "bad".

We have to pay attention to rename the variables, namely it is important, that the variables in the body of a unified rule let be different from the former unifications. To solve this problem, we will identify these variables by subscribing them with the level of the evaluation graph.

Let us attach labels to the edges of the form $Q \rightarrow R_1\theta, \ldots, R_n\theta$! Let the edge's label be the triplet $(\theta; I\beta)$.

Let the rule-body of the form $Q_1, \ldots, Q_n$ be a node of level $k = 2i + 1$! Then there is an n-order hyper-edge to the nodes $Q_1, \ldots, Q_n$. The hyper-edge has no label.

We can get an answer to the query from the labels of evaluating graph.

The path ending in the symbol "bad" doesn't give solution. Let us omit these paths! In other words, let us omit all of the edges and nodes which lead to this symbol independently from the fact, that these nodes are connected to each other by hyper-edges or ordinary edges. (If there is a path from one node of a hyper-edge to the symbol "bad", all of the nodes belonging to this hyper-edge and their descendants are cancelled.) The given graph is called searching graph.

A solution can be achieved along the path ending in the symbol "good" in the searching graph. The union of these solutions is the answer to the given query. The level of the atoms in the answer can be computed with the aid of the uncertainty-level function.

Definition 12 The function

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

is called uncertainty-level function.

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

$$f(I_1, \alpha, \beta) = \min(\alpha, \beta)$$

$$f(I_2, \alpha, \beta) = \max(0, \alpha + \beta - 1)$$

$$f(I_3, \alpha, \beta) = \alpha \cdot \beta$$

$$f(I_4, \alpha, \beta) = \begin{cases} 0 & \alpha + \beta \leq 1 \\ \beta & \alpha + \beta > 1 \end{cases}$$

$$f(I_5, \alpha, \beta) = \max(0, 1 + (\beta - 1)/\alpha), \alpha \neq 0$$

$$f(I_6, \alpha, \beta) = \alpha.$$

Let us determine the substitution θ along the hyper-path leading to the symbol "good" in the following way: (As a path contains hyper-edges, therefore the path may end in more leaves.)

For each hyper-node let us construct the join of the substitutions of the body's atoms. Let us order this joins to the nodes of even levels (that is to the nodes of the heads). Then let us construct the partial composition of these substitutions.

On answer to the query

$$(Q, \alpha)$$

is:

$$(Q\theta, \alpha_{goal}),$$

where α_{goal} can be computed recursively with the aid of uncertainty-level function $f(I, \alpha, \beta)$ in the following way:

Starting at the leaves, we order to them the value $\alpha = 1$, then we go backward to the root. If the uncertainty level of a node on the odd level of the graph is α, let the uncertainty level of the parent node be $\alpha = f(I, \alpha, \beta)$, 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 $\alpha = \min(\alpha_1, \ldots, \alpha_k)$. The uncertainty level of the root is α_{goal}.

Example 1 Let us see next rules:

$$p(a) \leftarrow; I_1; \beta_1$$

$$p(b) \leftarrow; I_2; \beta_2$$

$$r(c) \leftarrow; I_3; \beta_3$$

$$q(x, y) \leftarrow p(x), r(y); I_2; \beta_4$$

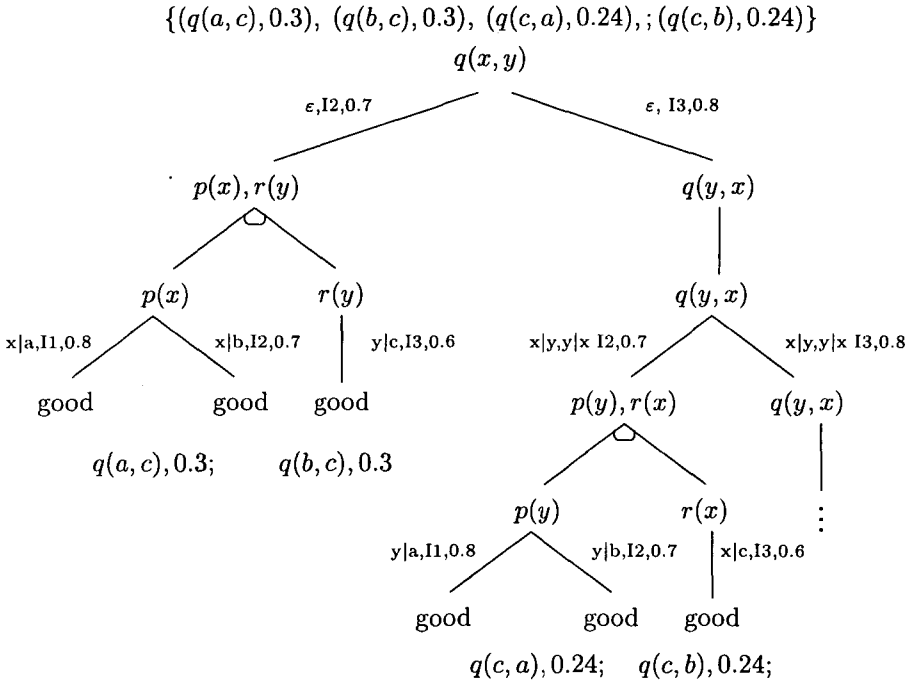
$$q(x, y) \leftarrow q(y, x); I_3; \beta_5$$

$$s(x) \leftarrow q(x, y); I_3; \beta_6$$

Let $\beta_1 = 0.8, \beta_2 = 0.7, \beta_3 = 0.6, \beta_4 = 0.7, \beta_5 = 0.8, \beta_6 = 0.9$

We want to determine $q(x, y)$.

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



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:

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.

Proof: We prove the equivalence of the two evaluations by induction on the depth of the evaluation graph.

Let us suppose that the depth of evaluation graph is one, that is all of the children of the root are the symbols "good" or "bad". This can occur only in the case if no rule's head can be unificated with the goal, or only facts can be unificated with that. In the first case, there is no answer to the query either in bottom-up, or top-down evaluations. In the second case, according to both of the evaluations, the answer is the same.

Let us suppose, that the theorem is true for all evaluating graphs, containing paths with length at least n.

Let us consider the evaluating graph, the maximum path-length of which is $n+1$.

Let us examine the sub-goals on the second level of the graph. The depth of the evaluation graph of these sub-goals is at least $n-1$, that is the induction assumption

is true. In bottom-up manner the goal can be reached only from these sub-goals. Going up to the first level in bottom-up manner along the hyper-edges, we get the bodies of the rules and the uncertainty level, from which we get the wanted answer. Applying the suitable substitution and computing the uncertainty factor, we get the same answer as in top-down manner.

Thus according to the induction hypothesis, the statement is true for all finite evaluation graphs. □

We give an algorithm to evaluate the graph. This algorithm provides the answer in the case of a given program and a given goal.

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_evalutaion" 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 with the aid of a selection function. The special symbols ("good", "bad") are not in the set of evaluable sub-goals, because they are not evaluable. (In the case of "bad" there is no a unificable rule, in the case of "good" 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 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 we can reduce the number of examinable paths.

During the evaluation of a sub-goal, it is possible to substitute such variables which don't appear among the variables of the sub-goal, therefore 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 with the aid of the set of substituting-terms. The set of substituting-terms of substitution $\theta = \{x_1|t_1, \ldots, x_n|t_n\}$ is the set $\{t_1, \ldots, t_n\}$.

Algorithm 4

```
Evaluation:
begin
      solution := ∅
      goalanswer := ∅
      goal_evaluation (goal, goalanswer)
      while not_empty (goalanswer) do
            (θ, αgoal) := element (goalanswer)
            goalanswer := goalanswer −{(θ, αgoal)}
            solution := solution ∪{( goal's_atom θ, αgoal )}
      endwhile
end
```

```
Procedure goal_evaluation (goal, goalanswer)
        goal_variables := { the set of the variables of the goal }
        R := {(r; I; β)| rule's_head (r) is unifiable with the goal }
        if R = ∅ then return
        while not_empty (R) do
                (r; I; β) := rule_selection (R)
                R := R − {(r; I; β)}
                body := rule's_body (r)
                for all variable ∈ r do
                        if variable ∈ substituting_terms (θ)
                                then variable := newname (variable)
                endfor
                θ := mgu(goal's_atom, rule's_head (r))
                body := body θ
                αbody := 1
                θ body := ε
                if body = ∅ then goalanswer := goalanswer ∪{θ, f(I, αbody, β))}
                  else rule_evaluation (body, αbody, θbody, goalanswer,
                          goal_variables, I, β)
                endif
        endwhile
endprocedure

Procedure rule_evaluation (body, αbody, θbody, goalanswer, goal_variables, I, β)
        atom := atom_selection (body)
        newbody := body −{ atom }
        answer := ∅
        goal_evaluation (atom, answer)
        if answer = ∅ then return
        while not_empty (answer) do
                (θ, αatom) := element (answer)
                answer := answer −{(θ, αatom )}
                θbody := θbodyθ
                αbody := min ( αbody, αatom)
                if newbody ≠ ∅ then
                  newbody := newbody θ
                  rule_evaluation (newbody, αbody, θbody, goalanswer,
                    goal_variables, I, β)
                endif
                if newbody = ∅ then
                        θ := projection (θbody, goal_variables)
                        goalanswer := goalanswer ∪{(θ, f(I, αbody, β))}
                endif
        endwhile
endprocedure
```

Note: The order of the unificable rules and sub-goals is unimportant, but it has an effect on the efficiency of the algorithm.

The uncertainty level of the goal $(Q;\alpha)$ is either constant or variable. If it is 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 those, whose uncertainty factors are greater then α. Thus, the size of the evaluation graph can be reduced.

As the above example shows, the top-down evaluation may not terminate. The reason is the evaluation of recursive atoms, because the evaluation of nonrecursive atoms terminates in finite steps. (The number of steps is $2t + 1$, where t is the longest path leading to the atom in the evaluating graph.)

If we order a depth limit to each recursive atom, the procedure can be stopped. This limit can be determined in the following way:

In a dependency graph let h be the maximum length of the loops containing the predicate of the atom, and let t be the maximum length of loop-free paths leading to this predicate.

Let us enter the concept of *recursion distance.* This is the number of steps in which we get the fixpoint respecting this atom in bottom-up evaluation. The recursion distance depends on the number of constant in the program and the "content" of the predicate.

For example in the case of the program

$$u(x,y) \leftarrow e(x,z), u(z,y); I; \beta_1$$

$$u(x,y) \leftarrow e(x,y); I; \beta_2$$

where e is a fact and c is the number of constant in the program, the recursion distance of atom $u(x,y)$ is $c-1$.

Let us denote the recursion distance by r! Then the depth limit is $k = 2h(r-1) + 2t + 1$.

Proposition 4 Let us order the previously defined depth limit to each atom of program P! Then the top-down evaluation terminates and gives all the solutions, which satisfies the goal.

Proof: As the goal-evaluation is driven back to the evaluation of the sub-goals, therefore it is enough to show the truth of the proposition for one recursive atom.

If there is no loop-free path to a rule's head-predicate in the dependency graph, the rule can not be evaluated. Thus, it is enough to look at the atoms to which there are loop-free paths.

If the length of a path leading to the predicate in the dependency graph is t, the length of this path in the evaluating graph is $2t$, because the evaluation graph is built from a series of two steps: determining the rule-bodies, and dividing them into sub-goals. There are additional edges leading to the ending symbols.

Along the $2t$ step-long path we get from the sub-goal to an atom of a fact-predicate, which can be evaluated.

The given atom - including other possible variables - occurs again in the evaluation graph in $2h$ steps deeper. The new evaluation is necessary only, if it provides a new solution. This possibility however can not occur more than the value of the recursion distance. As in the case of the first recurrence, we are on the second recursion level, so it is enough to allow the recurrence in $r - 1$ times.

As we don't get any new solution deeper then the given limit, we can leave these branches.

Ordering the suitable depth limit to each atom, the algorithm terminates, and gives all the solutions which satisfies the goal. □

Note: The recursion distance is not always as simple as in the example above, but it can not be greater then c^n, where c is the number of constants, n is the number of atoms in the program.

9 Top-down Evaluation in the Case of Stratified *f*DATALOG

It is easy to apply the top-down evaluation for stratified *f*DATALOG. In the case of stratified *f*DATALOG, 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 computed up to this point is α, let us continue the computation with value $1 - \alpha$.

10 Conclusion

In this article we have dealt with the evaluation of fuzzy DATALOG, given the algorithms of bottom-up and top-down evaluation, and showed the equivalence of two evaluations.

References

[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) 153-166.

[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), 346-353.

[K] Attila Kiss: On the least models of fuzzy Datalog programs, *International Conference on Information 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*, Research Studies Press LTD., Taunton, Somerset, England, 1990.

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

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

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

Received July, 1996