Retrieval strategy of fuzzy prolog LbFP

Hiroyuki Yasui* and Masao Mukaidono
*Musashi Institute of Technology
1-28-1 Tamatsutsumi, Setagaya-ku, Tokyo, 158-8557, Japan
Phone:+81-3-3703-3111, FAX:+81-3-5707-2169
e-mail:yasui@ipc.musashi-tech.ac.jp
Department of Computer Science, Meiji University
1-1-1 Higashimita, Tama-ku, Kawasaki, 214-8571, Japan
Phome:+81-44-934-7450, FAX:+81-44-934-7912
e-mail: masao@cs.meiji.ac.jp

ABSTRACT: A serious well-known problem of fuzzy prolog is that a same solution of a query has different proof paths and truth-values. It means that fuzzy prolog should retrieve the whole proof paths and will spend quite a long time for only one solution. Even the famous linear resolution (SDL resolution) is no enough for solving this problem. Accordingly, an efficient retrieval strategy is indispensable for fuzzy prolog.
In previous work we proposed a fuzzy prolog, named LbFP, adopted Lukasiewicz's implication for the implication operation and Lukasiewicz's product (bounded product) for the composition operation, respectively. LbFP is formulated in terms of tree resolution rather than the more usual linear resolution, because predicates of a rule body are combined using MIN operation which is different from the composition operation (Lukasiewicz's product) using in a derivation process.
In this paper, we prove our tree resolution's efficiency is equivalent to linear resolution's and show efficient retrieval strategy for LbFP.

KEYWORD: Fuzzy Prolog, Tree Resolution, Fuzzy Logic Programming

1. INTRODUCTION

A lot of prolog systems applied fuzzy theory, so called fuzzy prolog, has been studied. In previous works, we studied fuzzy logic programming theory by paying attention to the property of implication operation and composition operation on the inference process. By using two conditions about truth-value of fuzzy logical inference, we had proposed for fuzzy logic programming, we concluded that three pairs of operations were best for fuzzy logic programming. Our fuzzy prolog LbFP is adopted one of these pairs, Lukasiewicz's implication for the implication operation and Lukasiewicz's product (bounded product) for the composition operation, respectively Yasui (1994, 1995, 1996, 1998). In this section, we introduce our LbFP, which deals with fuzzy predicates, fuzzy horn clauses and crisp terms.

1.1. SYNTAX AND SEMANTICS

We consider a first-order language ***L*** that has a set of constants ***C***, a set of variables ***V***, a set of functors ***F*** and a set of predicates ***P***.

Definition 1: Truth-value
Let the closed interval [0,1] be the universe of truth-values of fuzzy logic.
Definition 2: Logical operations
Let a, b be variables of truth-value.
- *a ¬ b = min { 1, a-b+1 } (Lukasiewicz implication)*
- *a Ù b = min { a , b }*

Theorem 1: Term
*Let **x** Ì (**C** È **V**) and f Î **F**.*
*An arbitrary c Î **C**, v Î **V** and f(**x**) is called **term** and a set of terms is denoted by **T**.*

Definition 3: Fuzzy predicate

Let a fuzzy predicate p Î ***P*** *be a function from* $\mathbf{T}^n$ *into truth-values [0, 1].*
n is called ***arity*** *of p and p(****x****) is called an* ***atomic formula*** *and a set of atomic formulae is denoted by* ***U****.*

Definition 4: Fuzzy horn clause

Let B_1, ..., B_n, *and A be atomic formulae. A fuzzy horn clause is a fuzzy logical expression described by*

- *Rule* $A \leftarrow B_1, B_2, \ldots, B_n$
- *Fact* A

The first expression means $A \leftarrow (B_1 \wedge B_2 \wedge \ldots \wedge B_n)$ and A is called the **head** of rule and $B_1, B_2, \ldots, B_n$ is the **body** of rule. This expression is called **rule clause** and second expression is called **fact clause**.

Definition 5: Program

Let a set of fuzzy horn clauses attached truth values each other be called a ***program*** *and a fuzzy horn clause C in a program G be expressed* $m_G(C) = t$. *(in short* $m(C)$, *when no confusion)*

An attached truth-value τ means that a truth-value of a fuzzy horn clause is more than equals τ.

Definition 6: Substitution

Let an assignment of a term in ***U*** *to a variable term in terms of a atomic formula A be called a* ***substitution*** *and be denoted by Aq .*

Next, we discuss the inference rule of LbFP.

1.2. INFERENCE RULE

Proposition 1: Substitution

Let A be an atomic formula and q be a substitution.
Then Aq is a logical conclusion of the inference and the truth value of Aq is calculated as follows:
$m(Aq) = m(A)$

Proposition 2: Modus ponens

Let Aq be a logical conclusion of the inference from fuzzy horn clauses B' and A $\leftarrow$ *B where B'q = Bq for a substitution q. Then, the truth value of the conclusion is calculated as follows:*
$m(Aq) = m(B'q) * m(Aq \leftarrow Bq)$

Where * is the composition operation of truth values of premises and $a * b = max\{ a + b - 1 \}$ (Lukasiewicz product)

Proposition 3: Combination

Let $B_1, \ldots, B_n$ *be atomic formulae.*
Then the truth-value of the fuzzy expression $B_1 \wedge \ldots \wedge B_n$ *is calculated as follows:*
$m(B_1 \wedge \ldots \wedge B_n) = m(B_1) \wedge \ldots \wedge m(B_n)$

Where $\wedge$ is a fuzzy logical conjunction (*min*) operator.

Definition 7: Implicit fact

If G can be a logical conclusion of the inference, then it is an implicit fact, which plays the same role as any other real fact.

Proposition 4: *Tree resolution*

If a substitution q such that $B_1 q = B'_1 q, \ldots, B_n q = B'_n q$ *exists and* $B'_i q$ $(i = 1, \ldots, n)$ *are implicit facts, then the logical conclusion of* $B'_1, \ldots, B'_n$ *and* $A \leftarrow B_1, \ldots, B_n$ *is A q and* $m(A q) = m(A q \leftarrow B_1 q, \ldots, B_n q) * (m(B'_1 q) \wedge \ldots \wedge m(B'_n q))$

If we define the linear resolution method as in conventional logic programming theory then that combination rule doesn't need. However, we don't define it because the composition operation of inference differs from the logical conjunction operation in LbFP.
If the composition operation and the logical conjunction are the same then the truth-values of the answer by linear resolution and by tree resolution are also the same. But most composition operations (t-norm) of fuzzy logic are stronger than logical conjunction. Therefore, the truth-value of the answer by linear resolution is generally smaller than by tree resolution.

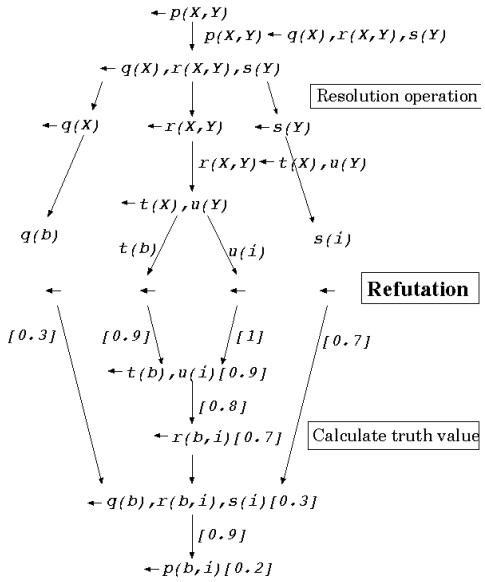

Figure 1: Illustration of the execution of p(X, Y)

The tree resolution can satisfy soundness and completeness under the constraint that only fact fuzzy horn clauses can be derived Yasui (1996).

Definition 8: Meaningless

A fuzzy horn clause whose truth-value is 0 is called ***meaningless****. If a logical conclusion is meaningless, then this logical conclusion is not available.*

1.3. REFUTATION

Refutation is the method used to prove whether knowledge is a logical conclusion of previously known knowledge in conventional logic programming. In fuzzy logic, generally we cannot use this refutation method, because it is based on the law of contradiction, which is not satisfied. But our LbFP, which adopts Lukasiewicz's product as composition operation, satisfies the law of contradiction.
Thus here, we can introduce the refutation method.

Definition 9: Goal clause

Let $G_1 \grave{U} ... \grave{U} G_n$ be a problem to be proved. Then, $\neg G_1, ..., G_n$ is called a ***goal clause****.*

Here $\neg G_1, ..., G_n$ means $0 \neg G_1 \grave{U} ... \grave{U} G_n$. Usually the *0* is omitted.

Theorem 2: Refutation method

Let a goal clause, $\neg G_1, ..., G_n$, be given. If $\neg$ is derived by tree resolution, then we call this a ***refutation*** *and the fuzzy expression $G_1 \grave{U} ... \grave{U} G_n$ is a logical conclusion of the program.*

$\neg$ is called an **empty clause** and a truth-value of this logical conclusion is same as $m(\neg)$ calculated by tree resolution.
The truth-value of the logical conclusion is called **refutation grade**.
The combination of the tree resolution and the refutation method corresponds to SDL resolution Mukaidono (1982).

Example 1: Example of the refutation method

Let a goal clause be $\neg p(X,Y)$ and the program be:

Knowledge	Truth value	Knowledge	Truth value
$p(X,Y) \neg q(X),r(X,Y),s(Y)$	0.9	$s(j)$	0.6
$r(X,Y) \neg t(X),u(Y)$	0.8	$t(a)$	0.2
$q(a)$	0.5	$t(b)$	0.9
$q(b)$	0.3	$u(i)$	1
$s(i)$	0.7	$u(j)$	0.8

Table I:. Example program

Figure 1 shows the refutation method.
The diagrams of the **Resolution operation** and of **Calculate truth-value** are symmetric with each other. Numbers within brackets show truth-values of corresponding clauses. For instance, $m(\neg t(b),u(i)) = 0.9$ is calculated from $min(m(t(b)), m(u(i))) = min(0.9, 1)$ and $m(\neg r(b, i)) = 0.7$ is from $m(\neg t(b),u(i)) * m(r(X,Y) \neg t(X),u(Y)) = 0.9 * 0.8$.
In the resolution operation, as $r(X,Y)$ is not a fact, $\neg r(X,Y)$ becomes the new goal clause.
Then, the results $p(b, i)$ and $p(b, j)$ are logical conclusions of the program and $m(q(b, i)) = 0.2$ and $m(p(b, j)) = 0.2$.
If we adopt a linear refutation method then the answer of this example cannot be proved, because the logical conclusions become meaningless immediately.

2. TREE RESOLUTION AND LINEAR RESOLUTION

Now we show our tree resolution's efficiency is equivalent to linear resolution's.

Linear resolution considers the body of a goal clause as a list of atomic formulae. And the first element of this list is replaced by a list corresponds to the body of a rule clause whom head is matched to this first element by a substitution. If the first element of this list is matched a fact, this element is deleted and the second becomes the first.
Similarly, tree resolution considers the body of a goal clause as a branch point of a tree and each atomic formula as each branch. And the branch of this tree is connected to a new branch point corresponds to the body of a rule clause whom head is matched to this branch by a substitution. If the branch of this tree is matched a fact, this branch is connected to a leaf.
Now tree resolution can be adopted two methods. The first method corresponds to the depth fast search algorithm and the second corresponds to the breadth fast algorithm.
Depth fast method is that the first branch of the tree is always connected to a new branch point. And the first branch of this new branch point becomes the new first branch of the tree. We adopted this method into our LbFP for more efficient retrieval strategy that is mentioned the next section.
Breadth fast method is that all branches of the tree are connected to new branch points each other.
If it considers a branch point of the depth fast tree resolution as a list of the linear resolution, it is obvious that the depth fast method is equivalent to linear resolution. Therefore, we can consider that our tree resolution's efficiency is equivalent to linear resolution's.

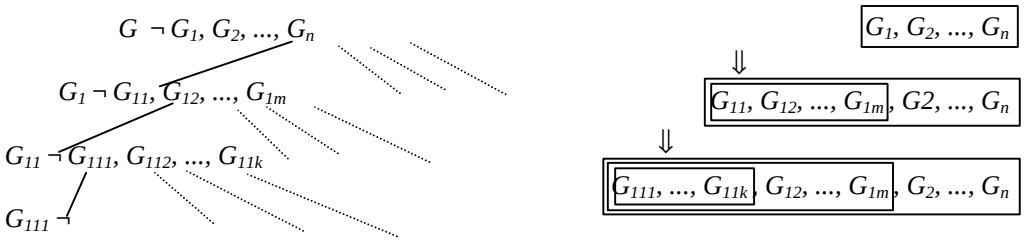

Figure 2: Illustration of tree resolution and linear resolution

3. RETRIEVAL STRATEGY

It is often the case that some different refutation processes for the same goal clause. Perhaps the first result of a refutation is t the best result. In order to know the greatest truth-value (refutation grade), we must try the whole different refutation processes of the same goal clause. Then the calculation time becomes very long, if there is no efficient retrieval strategy.
We consider that this problem is able to divide into two phases. The first phase is the retrieve for one result and the second is the retrieve for the best result.

3.1. FIRST PHASE

At fast, we have to find one result. In this case, the depth fast method may be equivalent to the breadth fast method. However, the depth fast method is more efficient than the breadth fast method by using the property of the meaningless, the truth-value becomes 0.
Assume that $\neg G, G' \neg G_1, \ldots, G_n$ and a substitution $G'q = Gq$ exist. The truth value of G is equal to the refutation grade and the refutation grade is calculated by: $m(Gq) = m(Gq \neg G_1q, \ldots, G_nq) * (m(G_1q) \dot{U} \ldots \dot{U} m(G_nq))$
Then, $t = m(Gq \neg G_1q, \ldots, G_nq)$

$$m(Gq) = t * (m(G_1q) \ldots, m(G_nq)) \pounds t * m(G_iq) \text{ If } t * m(G_iq) = 0$$

Then $m(Gq) \pounds 0$. Accordingly, this refutation path is failure and we have to search another rule clause matchable to the goal.
Operation *, $a * b = max\{ a + b - 1, 0\}$, is easy to become 0 on a cascade operation such as $a_1 * a_2 * \ldots * a_n$. It means that the more increase a number of * operations in a expression is, the smaller a truth value of this expression. On the other hand, operation $\dot{U}$ does not become 0 like this.
Therefore, the depth fast method is easier to be meaningless than the breadth fast method, because * operation is more often appeared in the depth fast than the breadth fast.
Our calculation algorithm of first phase is as follows:

Algorithm for one result

Let $\neg G$ be a depth i-th subgoal and t_j be a truth value attached to a rule clause corresponds to j-th branch point on the path from **root of the tree** to $\neg G$.

while clauses matchable to G exist:

- select a clause C matched to G.
- **if** $m(C) * t_1 * ... t_{i-1} = 0$ **then** C is rejected
- **if** $m(C) * t_1 * ... t_{i-1} > 0$ **then**
  - $t_i = m(C)$
  - the logical conclusion of G and C is $\neg G_1, ..., G_m$
  - **call** THIS ALGORITHM with each $\neg G_k$ as depth (i+1)-th subgoals.
  - $m(G) = t_i * min \{m(G_1), ..., m(G_m) \}$

end while

if $m(G)$ is not calculated **then** this subgoal $\neg G$ is rejected.

if $\neg G$ is O-th subgoal (real goal) **then** this refutation grade is $m(G)$.

3.2. SECOND PHASE

When we found one result with a refutation grade t, we can ignore another refutation path for the same goal clause such that it has a refutation grade less than equal t. And if we find other refutation with a refutation grade greater than t, this greater grade becomes next t.

Our calculation algorithm of second phase is as follows:

Algorithm for the best result

Let t be the best refutation grade until now.

while another refutation for the same goal clause $\neg G_0$ exist:

call DERIVATION with $\neg G_0$ as depth 0-th subgoal.

end while

procedure DERIVATION

Let $\neg G$ be a depth i-th subgoal and t_j be a truth value attached to a rule clause corresponds to j-th branch point on the path from root to $\leftarrow$ G.

while clauses whom truth value is better than t and matchable to G exist:

- select a clause C matched to G.
- **if** $m(C) * t_1 * ... t_{i-1} = 0$ **then** C is rejected
- **if** $m(C) * t_1 * ... t_{i-1} > t$ **then**
  - $t_i = m(C)$
  - the logical conclusion of G and C is $\neg G_1, ..., G_m$
  - **call** DERIVATION with $\neg G_k$ as depth (i+1)-th subgoals.
  - $m(G) = t_i * min \{m(G_1), ..., m(G_m) \}$

end while

if $m(G)$ is not calculated **then** this subgoal $\neg G$ is rejected.

if $\neg G$ is O-th subgoal (real goal) **then** the new best refutation grade is $m(G)$.

4. CONCLUSIONS

In this paper, we have displayed the tree resolution with depth fast method is equivalent to linear resolution and proposed efficient retrieval strategy for tree resolution.

REFERENCES

Dubois D.; Lang J.; Prade H., 1991, Fuzzy sets in approximate reasoning, Part1: Inference with possibility distributions. Fuzzy Sets and Systems 40, pp.143-202

Dubois D.; Lang J.; Prade H., 1991, Fuzzy sets in approximate reasoning, Part2: Logical approaches. Fuzzy Sets and Systems 40, pp.203-244

Van Emden M., 1986, Quantitative Deduction and its Fixpoint Theory. The journal of Logic Programming 4, 1, pp.37-53

Hindel C.J., 1986, Fuzzy Prolog. Int. Journal of Man-Machine Studies 24, pp.569-595

Ishizuka M.; Kanai N., 1985, Prolog-Elf incorporating fuzzy logic. Proc. of the 9th Int. Joint Conf. on Artificial Intelligence (IJCAI 85), Los Angels, pp.701-703

Kikuchi H., Mukaidono M., 1988, PROFIL: Fuzzy Interval Logic Prolog. Preprints of the Int. workshop on Fuzzy Systems Applications (IFSA), Iizuka, pp.205-206

Kikuchi H., Mukaidono M., 1994, Liner Resolution for Fuzzy Logic Program. Journal of Japan Society for Fuzzy Theory and Systems, Vol.6, No. 2, pp.294-303

Klawonn F.; Kruse R., 1994, A Lukasiewicz Logic Based Prolog., Mathware & Soft Computing 1, pp.5-29

Kowalski R., 1974, Predicate Logic as a Programming Language. Proc. of IFIP Congress. North-Holland, Amsterdam, pp.569-574

Kowalski R., 1979, Algorithm = Logic + Control. Comm. ACM 22, pp.424-436

Lee R.C.T., 1972, Fuzzy logic and the resolution principle. Journal of the Assoc. for Computing Machinery 19, pp.109-119

Lloyd J.W., 1987, Foundations of Logic Programming. Springer Verlag, Berlin

Martin T.P.; Baldwin J.F.; Pilsworth B.W., 1987, The implementation of FPROLOG - A fuzzy Prolog interpreter. Fuzzy Sets and Systems 23, pp.119-129

Mukaidono M., 1982, Fuzzy inference in resolution style. In: Fuzzy Sets and Possibility Theory - Recent Developments. Edited by R. R. Yager, Pergamon Press, pp.224-231

Robinson J.A., 1965, A Machine Oriented Logic Based on the Resolution Principle. Journal of ACM 12(1), pp.23-41

Shen Z.L.; Ding L.; Mukaidono M., 1988, A theoretical framework of fuzzy Prolog machine. In: Fuzzy Computing - Theory, Hardware and Applications (M. M. Gupta; T. Yamakawa, ed.). North-Holland, Amsterdam, pp.89-100

Sterling L.; Shapiro E., 1986, The Art of Prolog (2nd edition). MIT Press, Cambridge MA

Yasui H.; Mukaidono M., 1994, Postulates and proposals on Fuzzy Prolog. 2nd European Congress on Intelligent Techniques and Soft Computing (EUFIT '94), Aachen vol. 2, pp.1080-1086

Yasui H.; Hamada Y.; Mukaidono M., 1995, Fuzzy Prolog Based on L ukasiewicz Implication and bounded Product. Int. Joint Conf. of the 4th IEEE Int. Conf. on Fuzzy Systems and the 2nd Int. Fuzzy Engineering Symposium (FUZZ-IEEE/IFES95), vol. III , pp.949-955

Yasui H.; Mukaidono M., 1996, A Consideration on Fuzzy Logic Programming Based on Lukasiewicz's Implication. Journal of Japan Society for Fuzzy Theory and Systems, vol. 8, No. 5, pp.937-946

Yasui H.; Mukaidono M., 1998, Fuzzy Logic Programming Based on L ukasiewicz Implication. Chapter8, Logic Programming and Soft Computing, Research Studies Press, England, pp.147-162