First Int. Workshop on IFSs
Banska Bystrica, 22 Sept. 2005
NIFS 11 (2005), 6, 29-34

Intuitionistic Fuzzy Logical Programming. Syntax and Procedural semantics

K. Georgiev, T. Trifonov

Centre For Biomedical Engineering, Bulgarian Academy of Sciences,
Sofia, Bulgaria, "Acad. G. Bontchev" str, bl. 105,
k.georgiev@clbme.bas.bg, trifon@clbme.bas.bg

Abstract. This article opens the way for an implementation of Intuitionistic Fuzzy Logical Programming (IFLP) based on Intuitionistic Fuzzy Sets (IFS), as defined by Atanassov [1]. The proposed system suggests defining intuitionistic fuzzy certainty measures over programs clauses – facts and rules. An implementation of an IFL system would be a convenient tool for development of expert systems using uncertain or incomplete information and expert knowledge.

Keywords: Intuitionistic Fuzzy Logic, Fuzzy Prolog

1. Introduction

Logical programming is a widely applied and convenient paradigm for development of rule-based expert production systems. Implementations of an expert system based on uncertain or incomplete information needs an extension of the PROLOG language that uses fuzzy logic.

Existing researches on fuzzy logical programming explore the theoretical aspects of PROLOG language implementations based on fuzzy logic [2]. Such implementations allow fuzzification of the program facts or rules and produce the truth values of the derivations in the interval [0,1] or subintervals of [0,1], applying the laws of Fuzzy Logic or Interval Valued Fuzzy Logic, respectively.

This paper proposes a version of the logical programming capable of dealing with uncertain facts and rules, evaluated by independent degrees of truthfulness and falsity, i.e. intuitionistic fuzzy truth values. Intuitionistic fuzzy logic (IFL) [1] is chosen to be the underlying logical system of the Intuitionistic Fuzzy Logical Programming (IFLP). This work is a continuation and extension of previous researches on IFL bases logical programming.

The following sections propose a simple extension of the syntax of the PROLOG language and present the procedural semantics of Intuitionistic Fuzzy (IF) PROLOG. The purpose of this research is to open a way for an implementation of IF PROLOG interpreter.

2. Syntax of the Intuitionistic Fuzzy Logical Programs

This section proposes the syntax of IFL programs. We assume that the reader is familiar with the syntax of the PROLOG programming language.

As in the standard logical programming, an IFL program is comprised of *facts*, *rules,* and *queries*. The new element in the IFL programs, which opens the way of the Intuitionistic Fuzzy semantics, is the *certainty measure* of the database clauses - the facts and rules. In contrast to the standard logical programs, where all the clauses are absolutely true formulas, the certainty measure defines the IF truth value of the clauses of the IFL programs.

The certainty measure is then defined by the degree of truthfulness and the degree of untruthfulness of the clause, not necessarily summing up to 100%. We will write the certainty measure in square brackets ahead of all clauses of the program, omitting those only that are 100% true and 0% untrue. Let us consider the following program

```
                male(john).
                female(mary).
[0.90,0.10]     loves(john,wine).
[0.30,0.50]     loves(john,lemonade).

[0.80,0.20]     loves(X,Y):-male(X),female(Y).
[0.70,0.20]     dreams(X,Y):-loves(X,Y).
```

The program describes a world in which people dream about the objects of their affection in the 70% of the cases, does not in the 20% of the cases, and we cannot tell anything about 10% of the cases.

Let us assume that there are no two clauses in the program which depend on each other, i.e. every clause is a separate assertion. In classical logical programming clauses are connected by a conjunction. In order to resemble probabilistic reasoning, we will combine the certainty measures of the clauses by the IF multiplication operation (.), defined as follows:

$$\langle a_1, b_1 \rangle . \langle a_2, b_2 \rangle = \langle a_1 . a_2, b_1 + b_2 - b_1 . b_2 \rangle$$

However, we will regard the preconditions **within a single rule** connected by the ordinary IF conjunction ($\bigcap$):

$$\langle a_1, b_1 \rangle \bigcap \langle a_2, b_2 \rangle = \langle \min(a_1, a_2), \max(b_1, b_2) \rangle$$

Therefore, in this world it is 63% true and 28% untrue that John dreams about wine and only 56% true that John dreams about Mary. In this sense we would expect the query:

```
?-dreams(X,Y).
```

to produce the following results:

```
[0.63,0.28]     X=john,Y=wine;
[0.21,0.69]     X=john,Y=lemonade;
[0.56,0.36]     X=john,Y=mary;
```

The next sections discuss the semantics of IFL programs, or how the result and the certainty measures of the result are obtained.

3. Procedural semantics of the IFL programs. SLD resolution.

In this section we will present a resolution algorithm for IFL programs similar to the standard SLD resolution used in PROLOG. The new elements of the algorithm will be the means to obtain the certainty measure of the obtained result. We assume that the reader is familiar with the standard SLD resolution.

The purpose of this article is to briefly present a way for the implementation of an IFL language interpreter, so we will not dig deep into the theoretical aspects of IFL programming. We will propose the resolution algorithm by the means of IFL SLD tree, adopting definitions and theoretical results from researches on classical SLD resolution.

The clauses of an IFL program are a finite number of IF implications with the additional property of their certainty measure. We consider the clauses universally closed.

$$[\mu_i, \nu_i] H^i \leftarrow A^i{}_1, \ldots, A^i{}_{n_i}$$

where $i = 1,..n$ and **n** is the number of clauses of the program. H^i is called the clause *head* and $A^i{}_1, \ldots, A^i{}_{n_i}$ is called the clause *body*. The facts are special cases of clauses with empty (absolutely true) bodies.

An IFL program, like ordinary logical programs, is a finite conjunction of program clauses. The *query* is a disjunction of negative literals, or a clause with an empty (absolutely false) head. The process of obtaining the *result* of the program is the process of refuting the negation of the query, or *deriving* the *empty goal* from the program clauses and the negation of the query.

A *goal* is a conjunction of positive literals. Let $G = B_1, \ldots, B_k$ be a goal. A one-step SLD resolution of *G* and a program clause C_i is written as:

$$G \xrightarrow{C_i} G' \text{ by } \theta,$$

where C_i is a program clause renamed apart of *G* such that H_i and B_1 are unifiable, $G' = A^i{}_1, \ldots, A^i{}_{n_i} B_2, \ldots, B_k$ and θ is the most general unifier of B_1 and H_i.

The definition of an IFL SLD tree differs slightly from the definition of the standard SLD tree. The nodes of the IFL SLD tree are goals with certainty measures, attached to each of their literals. At the start of the resolution algorithm, these certainty measures are not yet known. They are calculated *after* the IFL SLD tree construction completes.

The first part of the resolution algorithm for a given query is like the standard algorithm:

1. At the start of the algorithm, the IFL SLD tree has a single node – its root, containing the negated query as a goal. Proceed with point 2 for the root of the IFL SLD tree.
2. Let $G = B_1, \ldots, B_k$ be a node of the IFL SLD tree that has no children. If *k=0* then the algorithm ***completes successfully***. If there are no program clauses such that their head and B_1 are unifiable, then the algorithm ***fails***.

Let there be k program clauses $C_{p_1},..,C_{p_k}$, $p_1 > ... > p_k$, such that $H_{p_1},..,H_{p_k}$ and B_1 are unifiable. For each of these clauses, calculate $G \xrightarrow{C_i} G'$ and θ and create a child of the tree node connecting it to the node by an arc, labeled by the corresponding clause and θ - the most general unifier of B_1 and the clause head. Set the goal of the child to $G'\theta$.

Proceed with point 2 for all the newly created children, from left to right. Erase those for which the algorithm fails.

The presented resolution algorithm creates the complete SLD tree which may be infinite – in this case the algorithm will not come to an end. The algorithms applied in common PROLOG implementations create only one child of a node at a time, trying another clause only if the selected clause fails the algorithm. They stop the first time an empty goal is achieved. However, these technical details are not in the focus of this paper.

The final result of the query is achieved by a traversal of the SLD tree by multiplying the unifiers that lay on the path from the tree root to some node with an empty goal. The next part of our algorithm needs to calculate the certainty measure of all the literals in the goal of the root of the IFL SLD tree.

Let us focus on a tree node with an empty goal. Let *h* be the height of the node, or the number of arcs on the path from the root to the node. The algorithm will perform *h* steps, *p=0,.,.h-1,* beginning from the node with empty goal and going up the IFL SLD tree. On each step the certainty measures of all the literals in the goal of the current tree node will be calculated.

At each step *p*, let $G = B_1,...,B_k$ be the goal in the current node's *parent*, C_i and θ be the clause and the unifier that label the arc between the parent and the node. We will provide that at each step, all of the uncertainty measures of the current node's literals are calculated by previous steps. Let us consider the following cases:

1. C_i is a fact. Then certainly $C_i = B_1\theta \leftarrow$. Set the certainty measure of B_1 to the certainty measure of the fact C_i.

If the node's parent is the root of the tree, then the algorithm completes, **stop**.
Proceed with the algorithm on the node's parent.

When the algorithm completes, all the literals in the IFL SLD tree root have their certainty measure calculated. All that needs to be done at this point is to calculate the certainty measure of the conjunction which produces the certainty measure of the query's answer.

4. Total IFL SLD resolution

In the previous section, we presented an algorithm, which calculates the certainty measure of a goal, which has been derived from an IFL program by an SLD resolution. However, a very important feature of the algorithm is the requirement for a fixed empty node in the SLD tree, from each the algorithm would start. Each such node corresponds to a single derivation of the query from the IFL program. Therefore, the certainty measures, which the algorithm computes need not be the same for each empty node, although the query is one and the same.

This peculiarity has a simple explanation: the derivation of the query is a proof; when the *reasons* used in the proof (the program clauses) are not fully truthful, then the result is also not fully truthful. Moreover, the more uncertain reasons we use, the more uncertain the result will be. Therefore, the result, calculated by the algorithm is not the certainty measure of the query, with respect to the program, but the certainty measure of the query *with respect to a given derivation* from the program.

A natural question rises: how should we define a certainty measure of the query *with respect to the whole IFL program*? This question is meaningful only for ground goals; for open goals different derivations may lead to different answer-substitutions. An intuitive answer is that we should consider the most reliable derivation of the query. The *sibling nodes* in the SLD tree are the alternatives for derivation using different clauses. Moreover, all sibling nodes are calculated by a one-step resolution with the first literal in the goal of their common parent. The algorithm for SLD resolution in the previous section will produce certainty measures for the goal for each sibling node. These certainty measures should be combined with an IF disjunction ($\bigcup$) to produce the *total certainty measure* of the parent node.

Here is a recursive algorithm for total IFL SLD resolution:

Starting from the root SLD node:

Let the current node is a goal $G = B_1,...,B_k$.

Set the certainty measures $[\mu_1,\nu_1],...,[\mu_k,\nu_k]$ of all literals in G to [0.0,1.0] (absolute falsity).

Consider each arc between the current node and its *child*, labeled C_i and θ (the clause and the unifier):

1. If C_i is a fact, then $C_i = B_1\theta \leftarrow$. and the child's goal is $G' = B_2,...,B_k$. Find recursively the certainty measures $[\mu_2',\nu_2'],...,[\mu_k',\nu_k']$ for *G'* and set $[\mu_1,\nu_1] = [\mu_1,\nu_1] \bigcup [\mu^i,\nu^i]$, where $[\mu^i,\nu^i]$ is the certainty measure of C_i and $[\mu_j,\nu_j] = [\mu_j,\nu_j] \bigcup [\mu_j',\nu_j']$ for j = 2,…k.
2. If $C_i = H^i \leftarrow A^i{}_1,...,A^i{}_{n_i}$ is a rule, then $B_1 = H^i$ and $G' = A^i{}_1,...,A^i{}_{n_i},B_2,...,B_k$ is the child's goal. Find recursively the certainty measures $[\mu_1'',\nu_1''],...,[\mu_n'',\nu_n''],[\mu_2',\nu_2'],...,[\mu_k',\nu_k']$ and set

$$[\mu_1,\nu_1] = [\mu_1,\nu_1] \bigcup [\mu^i,\nu^i].\left(\bigcap_{j=1}^{n}[\mu_j'',\nu_j'']\right),$$

where $[\mu^i,\nu^i]$ is the certainty measure of C_i and $[\mu_j,\nu_j] = [\mu_j,\nu_j] \bigcup [\mu_j',\nu_j']$ for j = 2,…k.

The algorithm described above performs depth-first traversal of the SLD tree. It finishes only if the tree is finite. However, for infinite trees with infinitely occurring empty nodes, for

which the obtained certainty measures converge, the total certainty measure of the query can also be defined. It would be an infinite disjunction of all certainty measures of the query with respect to the infinitely emerging empty nodes. Investigation of converging infinite IF PROLOG derivations will be a subject of further research.

5. Conclusions

Intuitionistic fuzzy logical programming (IFLP) is an extension of classical logical programming, which supports intuitionistic fuzzy certainty measures of the logical program facts and rules. The proposed procedural semantics provides a method for backward chaining based on SLD resolution using IF logical operations for estimation of the certainty measure of the derived answer of the query.

The research considers an aspect of IFLP, which is not present in classical logical programming: the certainty measure of a ground goal with respect to the whole program, as opposed to a fixed derivation.

The Fixed-point semantics of IFLP is a target of future works on the topic, as well as the implementation of an IFLP interpreter. An implementation of an IFL system would be a convenient tool for development of expert systems using uncertain or incomplete information and expert knowledge.

References

[1] Atanassov, K. Intuitionistic Fuzzy Sets. Springer-Physica Verlag, Heidelberg, 1999.

[2] Claudio Vaucheret, Sergio Guadarrama, Susana Muñoz, **Fuzzy Prolog: A simple implementation using CLP (*R*),** International Conference in Logic Programming, ICLP 2002. Springer. LNCS 2401, pp. 469. Copenague, Denmark.