Fuzzy Prolog:
A Simple Implementation using CLP($\mathcal{R}$)

Claudio Vaucheret[1], Sergio Guadarrama[1], and Susana Muñoz[2]

[1] Departamento de Inteligencia Artificial
`claudio@clip.dia.fi.upm.es`
`sguada@isys.fi.upm.es`
[2] Departamento de Lenguajes de Programación y Sistemas
`susana@fi.upm.es`***
Universidad Politécnica de Madrid
28660 Madrid, Spain

Abstract. We present a definition of a Fuzzy Prolog Language that models interval-valued Fuzzy logic, and subsumes other fuzzy prologs. We give the declarative and procedural semantics for fuzzy logic programs. In addition, we give an implementation of an interpreter for this language made using CLP($\mathcal{R}$). We have incorporated uncertainty into a Prolog system in a simple way thanks to this constrains system. The implementation is based on syntactic expansion of the source code running on Prolog.

Keywords Fuzzy Prolog, Modeling Uncertainty, Logic Programming, Constraint Programming Application, Implementation of Fuzzy Prolog.

1 Introduction

The result of introducing Fuzzy Logic into Logic Programming has been the development of several "Fuzzy Prolog" systems. These systems replace the inference mechanism of Prolog by a fuzzy variant which is able to handle partial truth. Most of these systems implement the fuzzy resolution introduced by Lee in [Lee72], examples are the Prolog-Elf system [IK85], FPROLOG system [BMP95] and the f-prolog language [LL90]. However, there was no common way for fuzzifying Prolog as it has been noted in [SDM89]. Some of these Fuzzy Prolog systems only consider the fuzziness of predicates whereas other systems consider fuzzy facts or fuzzy rules. There is no agreement about which fuzzy logic must be used. Most of them use min-max logic (for modeling the conjunction and disjunction operations) but other systems use Lukasiewicz logic [KK94]. In this paper, we propose another approach that is more general in two aspects:

1. Truth value will be a sub-interval on $[0, 1]$. In fact, it could be a finite union of sub-intervals as we will see below. Having a unique truth value is a particular case modeled with a unitary interval.

*** contact author

2. Truth value will be propagated through the rules by means of a set of *aggregation operators*. The definition of an *aggregation operator* is a generalization that subsumes conjunctive operators (triangular norms as min, prod, etc), disjunctive operators (triangular co-norms as max, sum, etc), average operators (averages as arithmetic average, cuasi-linear average, etc) and hybrid operators (combinations of previous operators).

We add uncertainty to a Prolog compiler using CLP($\mathcal{R}$) instead of implementing a new fuzzy resolution as other fuzzy prologs. In this way, we use the original inference mechanism of Prolog, and we use the constraints and its operations provided by CLP($\mathcal{R}$) to handle the concept of partial truth. We represent intervals as constrains over real numbers and *aggregation operators* as operations with constraints.

The goal of this paper is to show how the Prolog inference system combined with Fuzzy Logic can produce a powerful tool to solve complex problems that contains uncertainty, and to present an implementation of a Fuzzy Prolog like a natural application of CLP($\mathcal{R}$)

The rest of the paper is organized as follows. Section 2 describes the syntax and the multi-valued semantics of our fuzzy system. Section 3 gives details about the implementation using CLP($\mathcal{R}$). Finally, we conclude and discuss some future work (Section 4).

2 Syntax and Semantics

In this section we present both the language and its semantics for our Fuzzy Prolog system. Firstly we generalize the concept of truth value of a predicate considering partial truth. Secondly we present the syntax and the different semantics and finally we illustrate the language with some examples.

2.1 Truth value

Given a relevant universal set X, any arbitrary fuzzy set A is defined by a function $A : X \rightarrow [0, 1]$ unlike the crisp set that is defined by a function $A : X \rightarrow \{0, 1\}$. This definition of fuzzy set is by far the most common in the literature as well as in the various successful applications of the fuzzy set theory. However, several more general definitions of fuzzy sets have also been proposed in the literature. The primary reason for generalizing ordinary fuzzy sets is that their membership functions are often overly precise. They require to assign a particular real number to each element of the universal set. However, for some concepts and contexts, we may only be able to identify approximately appropriate membership functions. An option is considering a membership function which does not assign to each element of the universal set one real number, but an interval of real numbers. Fuzzy sets defined by membership functions of this type are called *interval-valued fuzzy sets* [KY95,NW00]. These sets are defined formally by functions of the form $A : X \rightarrow \mathcal{E}([0, 1])$. Where $\mathcal{E}([0, 1])$ denotes the family of all closed intervals of real numbers in $[0, 1]$.

In this paper we propose to generalize this to have membership functions which assign to each element of the universal set one element of the Borel Algebra over the interval $[0,1]$. These sets are defined by functions of the form $A : X \rightarrow \mathcal{B}([0,1])$, where an element in $\mathcal{B}([0,1])$ is a countable union of sub-intervals of $[0,1]$.

The truth value of a predicate will depend on the value of other predicates which are in its definition. We use *aggregation operators* [Pra99] in order to propagate the truth value by means of the fuzzy rules. Fuzzy sets *aggregation* is made by the application of a numeric operator of the form $f : [0,1]^n \rightarrow [0,1]$. If it verifies $f(0,\ldots,0) = 0$ and $f(1,\ldots,1) = 1$, and in addition it is monotonic and continuous, then it is called *aggregation operator*. Dubois, in [DP85], proposes a classification of these operators with respect to their behavior in three groups:

1. *Conjunctive Operators* (less or equal to *min*), for example T-norms
2. *Disjunctive Operators*, (greater or equal to *max*), for example T-conorms
3. *Average Operators* (between *min* and *max*)

There is a need for a generalization of *aggregation operator* of numbers to *aggregation operator* of intervals. Following the theorem proven by Nguyen and Walker in [NW00] to extend t-norms and t-conorms to intervals, we propose the next definitions.

Definition 1 (F-aggregation). *Given an f-aggregation $f : [0,1]^n \rightarrow [0,1]$ an F-interval-aggregation $F : \mathcal{E}([0,1])^n \rightarrow \mathcal{E}([0,1])$ is defined as follows:*

$$F([x_1^l, x_1^u], ..., [x_n^l, x_n^u]) = [f(x_1^l, ..., x_n^l), f(x_1^u, ..., x_n^u)]$$

Definition 2 ($\mathcal{F}$-aggregation). *Given an F-aggregation $F : \mathcal{E}([0,1])^n \rightarrow \mathcal{E}([0,1])$ defined over intervals, an $\mathcal{F}$-aggregation $\mathcal{F} : \mathcal{B}([0,1])^n \rightarrow \mathcal{B}([0,1])$ defined over union of intervals as follows:*

$$\mathcal{F}(B_1, \ldots, B_n) = \cup\{F(\mathcal{E}_1, ..., \mathcal{E}_n) \mid \mathcal{E}_i \in B_i\}$$

In the presentation of the theory of possibility [Zad78], Zadeh considers that fuzzy sets act as an elastic constraint on the values of a variable and fuzzy inference as constraint propagation.

In our approach, truth values and the result of aggregations will be represented by constraints. A constraint is a Σ-formula where Σ is a signature that contains the real numbers, the binary function symbols $+$ and $*$, and the binary predicate symbols $=$, $<$ and $\leq$. If the constraint c has solution in the domain of real numbers in the interval $[0,1]$ then we say c is consistent, and we denote it as $[0,1] \models c$.

2.2 The Language

The alphabet of our language consists of the following kinds of symbols: *variables*, *constants*, *function symbols* and *predicate symbols*. A *term* is defined inductively as follows:

1. A *variable* is a *term*.
2. A *constant* is a *term*.
3. if f is an n-ary *function symbol* and $t_1, \ldots, t_n$ are *terms*, then $f(t_1, \ldots, t_n)$ is a *term*.

If p is an n-ary *predicate symbol*, and $t_1, \ldots, t_n$ are *terms*, then $p(t_1, \ldots, t_n)$ is an *atomic formula* or, more simply an *atom*.

A *fuzzy program* is a finite set of *fuzzy facts*, and *fuzzy clauses*. They are defined below.

Definition 3 (fuzzy fact). *If A is an atom,*

$$A \leftarrow v$$

is a fuzzy fact*, where v, a truth value, is an element in $\mathcal{E}([0,1])$ expressed as constraints over the domain $[0,1]$.*

Definition 4 (fuzzy clause). *Let $A, B_1, \ldots, B_n$ be atoms. A* fuzzy clause *is a clause of the form*

$$A \leftarrow_F B_1, \ldots, B_n$$

where F is an aggregation operator *of truth values represented as constraints over the domain $[0,1]$. It is an f-aggregation which induces an F-aggregation as by definition 1*

Definition 5 (fuzzy query). *A* fuzzy query *is a tuple*

$$v \leftarrow A\ ?$$

where A is an atom, and v is a variable (possibly instantiated) that represents a truth value.

2.3 Semantics

The *Herbrand Universe* U is the set of all ground *terms*, which can be made up of the constants and function symbols of the language, and the *Herbrand Base* B is the set of all ground atoms which can be formed by using predicate symbols (of the language) with ground *terms* (of the *Herbrand Universe*) as arguments.

Definition 6 (interpretation). *An* interpretation *I consists of the following:*

1. *a subset B_I of the* Herbrand Base*,*
2. *a mapping V_I, which assigns a truth value to each element of B_I.*

The Borel Algebra $\mathcal{B}([0,1])$ is a complete lattice under interval inclusion and the Herbrand Base is a complete lattice under set inclusion, therefore a set of *interpretations* forms a complete lattice under the relation $\sqsubseteq$ defined as follows.

Definition 7 (relation $\sqsubseteq$). *$I \sqsubseteq I'$ if and only if $B_I \subseteq B_{I'}$ and for all $B \in B_I$, $V_I(B) \subseteq V_{I'}(B)$.*

Definition 8 (model). *Given an* interpretation $I = \langle B_I, V_I \rangle$

- *I is a* model *for a* fuzzy fact $A \leftarrow v$*, if* $A \subseteq B_I$ *and* $v \subseteq V_I(A)$.
- *I is a* model *for a clause* $A \leftarrow_F B_1, \ldots, B_n$ *when the following holds: if* $B_i \subseteq B_I$, $1 \leq i \leq n$, *and* $v = \mathcal{F}(V_I(B_1), \ldots, V_I(B_n))$ *then* $A \subseteq B_I$ *and* $v \subseteq V_I(A)$
- *I is a* model *of a* fuzzy program, *if it is a* model *for the facts and clauses of the program.*

The least *model* of a program P under the $\sqsubseteq$ ordering, denoted by $lm(P)$, is called the *meaning* of the program P.

2.4 FixedPoint Semantics

The fixedpoint semantics we present is based on a one-step consequence operator T_P. The least fixedpoint I such that $T_P(I) = I$ is the declarative meaning of the program P and is equal to $lm(P)$.

Let P a fuzzy definite program and B_P the Herbrand base of P, the *mapping* T_P over *interpretations* is defined as follows:

Let $I = \langle B_I, V_I \rangle$ be a fuzzy *interpretation*, then $T_P(I) = I'$, $I' = \langle B_{I'}, V_{I'} \rangle$

$$
\begin{aligned}
B_{I'} = \{A \in B_P : \; & A \leftarrow v \text{ is a ground instance of a fact in } P \text{ and } [0,1] \models v \\
& \text{or} \\
& A \leftarrow_F A_1, \ldots, A_n \text{ is a ground instance of a clause in } P, \\
& \{A_1, \ldots, A_n\} \subseteq B_I \\
& v = \mathcal{F}(V_I(A_1), \ldots, V_I(A_n)), \\
& \text{and } [0,1] \models v\}
\end{aligned}
$$

$$
\begin{aligned}
V_{I'}(A) = \bigcup\{v \in \mathcal{B}([0,1]) : \; & A \leftarrow v \text{ is a ground instance of a fact in } P \text{ and } [0,1] \models v \\
& \text{or} \\
& A \leftarrow_F A_1, \ldots, A_n \text{ is a ground instance of a clause in } P, \\
& \{A_1, \ldots, A_n\} \subseteq B_I \\
& v = \mathcal{F}(V_I(A_1), \ldots, V_I(A_n)), \\
& \text{and } [0,1] \models v\}
\end{aligned}
$$

2.5 Operational Semantics

The procedural semantics is interpreted as a sequence of transitions between different states of the system. We represent the state of a *transition system* in a computation as a tuple $\langle A, \sigma, S \rangle$ where A is the goal, σ is a substitution representing the instantiation of variables needed to get this state and S is a constraint that represents the truth value of the goal at this state.

When computation starts, the first argument of the first state is the goal and the other arguments are the empty sustitution and the constraint of the query. When we get a state where the first argument is empty then we have finished the computation and the other two arguments represent the answer.

A *transition* in the *transition system* is defined as:

– $\langle A \cup a, \sigma, S\rangle \rightarrow \langle A\theta, \sigma \cdot \theta, S \wedge \mu_a = v\rangle$
  if $h \leftarrow v$ is a fact of the program P, θ is the mgu of a and h, and μ_a is the truth variable for a, and $[0,1] \models S \wedge \mu_a = v$
– $\langle A \cup a, \sigma, S\rangle \rightarrow \langle (A \cup B)\theta, \sigma \cdot \theta, S \wedge c\rangle$
  if $h \leftarrow_F B$ is a rule of the program P, θ is the mgu of a and h, c is the constraint for aggregator F on the truth variables for B, and $[0,1] \models S \wedge c$
– $\langle A \cup a, \sigma, S\rangle \rightarrow fail$
  if none of the above are applicable.

The success set $SS(P)$ collects the answers to simple goals $p(\widehat{x})$. It is defined as follows:

$SS(P) = \langle B, V\rangle$ where $B = \{p(\widehat{x})\sigma | \langle p(\widehat{x}), \epsilon, true\rangle \rightarrow^* \langle \emptyset, \sigma, S\rangle\}$ and $V(p(\widehat{x})) = \cup\{v | \langle p(\widehat{x}), \epsilon, true\rangle \rightarrow^* \langle \emptyset, \sigma, S\rangle$, and v is the solution of $S\}$

The three semantics are equivalent, i.e we have $SS(P) = lfp(TP) = lm(P)$.

2.6 Example

Suppose we have the following program

$tall(john) \leftarrow 0.8$
$swift(john) \leftarrow 0.7$
$good\_player(X) \leftarrow_{luka} tall(X), swift(X)$

Here, we have two facts, $tall(john)$ and $swift(john)$ whose truth values are the unitary intervals $[0.8, 0.8]$ and $[0.7, 0.7]$ respectively and a clause for the *good_player* predicate whose *aggregation operator* is the Lukasiewicz T-norm.

Consider the fuzzy goal

$$\mu \leftarrow good\_player(X) \quad ?$$

the first *transition* in the computation is

$\langle\{(good\_player(X)\}, \epsilon, true\rangle \rightarrow$
$\langle\{tall(X), swift(X)\}, \epsilon, \mu = max(0, \mu_{tall} + \mu_{swift} - 1)\rangle$

unifying the goal with the clause and adding the constraint corresponding to Lukasiewicz T-norm. The next *transition* leads to the state:

$$\langle\{swift(X)\}, \{X = john\}, \mu = max(0, \mu_{tall} + \mu_{swift} - 1) \wedge \mu_{tall} = 0.8\rangle$$

after unifying $tall(X)$ with $tall(john)$ and adding the constraint regarding the truth value of the fact. The computation ends with:

$$\langle\{\}, \{X = john\}, \mu = max(0, \mu_{tall} + \mu_{swift} - 1) \wedge \mu_{tall} = 0.8 \wedge \mu_{swift} = 0.7\rangle$$

As $\mu = max(0, \mu_{tall} + \mu_{swift} - 1) \wedge \mu_{tall} = 0.8 \wedge \mu_{swift} = 0.7$ entails $\mu = 0.5$, the answer to the query $good\_player(X)$ is $X = john$ with truth value 0.5.

3 Syntax and Implementation: CLP($\mathcal{R}$)

We decided to implement this interpreter as a syntactic extension of a CLP($\mathcal{R}$) system. This syntactic expansion was incorporated as a library in the Ciao Prolog system[1]. Constraint Logic Programming [JL87] began as a natural merger of two declarative paradigms: constraint solving and logic programming. This combination helps make CLP programs both expressive and flexible, and in some cases, more efficient than other kinds of logic programs. CLP($\mathcal{R}$) [JMSY92] has linear arithmetic constraints and computes over the real numbers.

Ciao Prolog is a next-generation logic programming system which, among other features, has been designed with modular incremental compilation in mind. Its module system [CH00] will allow to have classical modules and fuzzy modules in the same program and it incorporates CLP($\mathcal{R}$).

Many Prolog systems have included the possibility of changing or expanding the syntax of the source code. One way is using the `op/3` builtin and another is defining *expansions* of the source code by allowing the user to define a predicate typically called `term_expansion/2`. Ciao has redesigned these features so that it is possible to define source translations and operators that are local to the module or user file defining them. Another advantage of the module system of Ciao is that it allows separating code that will be used at compilation time from code which will be used at run-time.

We have written a library (or *package* in the Ciao Prolog terminology) called *fuzzy* which implements the interpreter of our fuzzy prolog language described in section 2.

Each fuzzy predicate has an additional argument which represents its truth value. A fact $A \leftarrow v$ is represented by a Fuzzy Prolog fact whose last argument stores the truth value v as a constraint over $\mathcal{R}$. The following examples illustrate the concrete syntax of programs:

$tall(john) \leftarrow [0.8, 0.9]$	`tall(john,M) :~ M .>=. 0.8, M .=<. 0.9`
$good\_player(X) \leftarrow_{min} tall(X), swift(X)$	`good_player(X,M) :~ min tall(X,M1), swift(X,M2).`

These clauses are expanded at compilation time to constrained clauses that are managed by CLP($\mathcal{R}$) at run-time. For example,

```
p(X,Mp) :~ min q(X,Mq),r(X,Mr).
```

is expanded to

```
p(X,Mp) :- q(X,Mq),r(X,Mr),
           minim([Mq,Mr],Mp),
           Mp .>=.0,Mp .=<.1.
```

[1] The Ciao system including our Fuzzy Prolog implementation can be downloaded from http://www.clip.dia.fi.upm.es/Software.

The predicate minim/2 is included as run-time code by the library. Its function is adding constraints to the truth value variables in order to implement the T-norm *min*.

```
minim([],_).
minim([X],X).
minim([X,Y|Rest],Min):-
        min(X,Y,M),
        minim([M|Rest],Min).
```

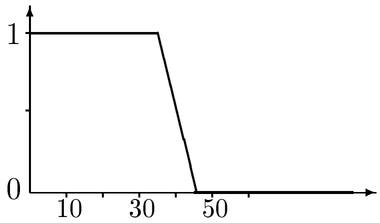
```
min(X,Y,Z):- X .=<. Y , Z .=. X.
min(X,Y,Z):- X .>. Y, Z .=. Y .
```

We have implemented several *aggregation operators* as prod, max,luka, etc. in a similar way and any other operator can be added to the system without any effort. The system is extensible by the user simply adding the code for new *aggregation operators* to the library.

Fuzzy predicates with piecewise linear continous membership functions like young/2 in figure 1 are written by:

```
young :# fuzzy_predicate([(0,1),(35,1),(45,0),(120,0)]).
```

This friendly syntax is translated to arithmetic constraints. We can even define the predicate directly if we prefer it. The translation is the following:




```
young(X,1):-     X .>=. 0,
                 X .<. 35.
young(X,M):-     X .>=. 35,
                 X .<. 45,
                 10*M .=. 45-X.
young(X,0):-     X .>=. 45,
                 X .=<. 120.
```

Fig. 1. fuzzy predicate young/2

It is possible to fuzzify crisp predicates. For example to fuzzify $p/2$ it is only necessary to write:

```
p_f :# fuzzy p/2
```

and the program is expanded with a new fuzzy predicate $p\_f/3$ (the last argument is the truth value) with truth value equal to 0 if $p/2$ fail and 1 otherwise.

We provide too the possibility of having the predicate that is the fuzzy negation of a fuzzy predicate. For this predicate $p\_f/3$ we will define a new fuzzy predicate called for example $notp\_f/3$ with the following line:

```
notp_f :# fnot  p_f/3
```

that is expanded at compilation time as:

```
notp_f(X,Y,M) :-
       p_f(X,Y,Mp),
       M .=. 1 - Mp.
```

3.1 Example

A simple example could be trying to measure which is the possibility that a couple of values, obtained throwing two loaded dice, sum 5. Let us suppose we only know that one die is loaded to obtain a small value and the other is loaded to obtain a large value. *small* and *large* are fuzzy concepts and we represent them by the following fuzzy predicates:

```
small :# fuzzy_predicate([(1,1),(2,1),(3,0.7),(4,0.3),(5,0),(6,0)]).
large :# fuzzy_predicate([(1,0),(2,0),(3,0.3),(4,0.7),(5,1),(6,1)]).
```

In fuzzy prolog this problem can be represented using min-max logic or other T-norm and T-conorm as `prod` and `dprod` as in the following two programs:

```
die1(X,M) :~ min
     small(X,M).

die2(X,M) :~ min
     large(X,M).

two_dice(X,Y,M):~ min
     die1(X,M1),
     die2(Y,M2).

sum(5,M) :~ max
     two_dice(4,1,M1),
     two_dice(1,4,M2),
     two_dice(3,2,M3),
     two_dice(2,3,M4).

  ?- sum(5,M).
.=.(M,0.7) ?
yes
```

```
die1(X,M) :~ prod
     small(X,M).

die2(X,M) :~ prod
     large(X,M).

two_dice(X,Y,M):~ prod
     die1(X,M1),
     die2(Y,M2).

sum(5,M) :~ dprod
     two_dice(4,1,M1),
     two_dice(1,4,M2),
     two_dice(3,2,M3),
     two_dice(2,3,M4).

  ?- sum(5,M).
.=.(M,0.79) ?
yes
```

`two_dice(X,Y,M)` represents the possibility that the first die gives `X` and at the same time the second die gives `Y`. `sum(5,M)` aggregates the possibilities of the four cases in which the two dice can sum 5.

4 Conclusions and Future work

The novelty of the Fuzzy Prolog presented is that it is implemented over Prolog instead of implementing a new resolution system. This gives it a good potential for efficiency, more simplicity and flexibility. For example *aggregation operators* can be added with almost no effort. The way of doing this extension to Prolog is interpreting fuzzy reasoning as a set of constraints [Zad78] and after that translating fuzzy predicates into CLP($\mathcal{R}$) clauses. The rest of the computation is resolved by the compiler.

Most of the other Fuzzy Prolog consider only one operator to get the truth value of the fuzzy clauses. We have generalized all operators with the concept of aggregation and this makes our Fuzzy Prolog subsume all the ways of resolution of the others. Another advantage of our approach is that it can be implemented with little effort over any other CLP($\mathcal{R}$) system.

Actually we are solving some semantic problems that arise when we try to combine crisp and fuzzy logic in the same programming language and we hope to present our result in this field soon.

We are also working now on the optimization of this system trying to improve the possibility to obtain constructive answers at any case. Another direction to continue with this work is to implant the expansion over other CLP($\mathcal{R}$) systems.

Acknowledgement

Authors would like to thank the suggestions of Enric Trillas and Francisco Bueno to improve the content and the ideas behind the paper.

References

[BMP95] J. F. Baldwin, T.P. Martin, and B.W. Pilsworth. *Fril: Fuzzy and Evidential Reasoning in Artificial Intelligence*. John Wiley & Sons, 1995.

[CH00] D. Cabeza and M. Hermenegildo. A New Module System for Prolog. In *International Conference on Computational Logic, CL2000*, LNCS. Springer-Verlag, July 2000. To appear.

[DP85] D. Dubois and H. Prade. A review of fuzzy set aggregation connectives. *Information Sciences*, 36:85–121, 1985.

[IK85] Mitsuru Ishizuka and Naoki Kanai. Prolog-ELF incorporating fuzzy logic. In *IJCAI*, pages 701–703, 1985.

[JL87] J. Jaffar and J.-L. Lassez. Constraint Logic Programming. In *ACM Symp. Principles of Programming Languages*, pages 111–119. ACM, 1987.

[JMSY92] J. Jaffar, S. Michaylov, P.J. Stuckey, and R.H.C. Yap. The clp(∇) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, 1992.

[KK94] Frank Klawonn and Rudolf Kruse. A Łukasiewicz logic based Prolog. *Mathware & Soft Computing*, 1(1):5–29, 1994.

[KY95] G. J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic*. Prentice Hall, 1995.

[Lee72] R.C.T. Lee. Fuzzy logic and the resolution principle. *Journal of the Association for Computing Machinery*, 19(1):119–129, 1972.

[LL90] Deyi Li and Dongbo Liu. *A Fuzzy Prolog Database System*. John Wiley & Sons, New York, 1990.

[NW00] H. T. Nguyen and E. A. Walker. *A first Course in Fuzzy Logic*. Chapman & Hall/Crc, 2000.

[Pra99] A. Pradera. *A contribution to the study of information aggregation in a fuzzy environment*. PhD thesis, Technical University of Madrid, 1999.

[SDM89] Z. Shen, L. Ding, and M. Mukaidono. Fuzzy resolution principle. In *Proc. of 18th International Symposium on Multiple-valued Logic*, volume 5, 1989.

[Zad78] L. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy sets and systems*, 1(1):3–28, 1978.