# Fuzzy Prolog:
## A Simple General 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, Sistemas de la Información
e Ingeniería del Software
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 former approaches because it uses a truth value representation based on a union of intervals of real numbers and it is defined using general operators that can model different logics. We give the declarative and procedural semantics for Fuzzy Logic programs. In addition, we present the implementation of an interpreter for this language conceived using CLP($\mathcal{R}$). We have incorporated uncertainty into a Prolog system in a simple way thanks to this constraints system. The implementation is based on a syntactic expansion of the source code during the Prolog compilation.

**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 with a fuzzy variant which is able to handle partial truth. Most of these systems implement the fuzzy resolution introduced by Lee in [Lee72], examples being the Prolog-Elf system [IK85], Fril Prolog system [BMP95] and the F-Prolog language [LL90]. However, there was no common method 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 just use Lukasiewicz logic [KK94].

There is also an extension of constraint logic programming [BMR01], which can model logics based on semiring structures. This framework can models the min-max fuzzy logic that is the only one with semiring structure.

On the other hand in [Sha83] logic programming is considered a useful tool for implementing methods for reasoning with uncertainty.

In this paper, we propose another approach that is more general in two aspects:

1. A truth value will be a finite union of sub-intervals on $[0, 1]$. An interval is a particular case of union of one element, and a unique truth value is a particular case of having an interval but with only one element.
2. A truth value will be propagated through the rules by means of an *aggregation operator*. The definition of *aggregation operator* is general. It subsumes conjunctive operators (triangular norms, [KMP], as min, prod, etc), disjunctive operators (triangular co-norms as max, sum, etc), average operators (averages as arithmetic average, quasi-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 do. In this way, we use the builting inference mechanism of Prolog, the constraints and their operations provided by CLP($\mathcal{R}$) to handle the concept of partial truth. We represent intervals as constraints over real numbers and *aggregation operators* as operations with these constraints.

We have found, e.g. in [NW00], an interpretation of truth values as intervals but we are proponing for the first time to generalize this concept to union of intervals. We will talk about their utility below.

The goal of this paper is to show how introducing fuzzy reasoning in a Prolog system can produce a powerful tool to solve complex uncertainty problems and to present an implementation of a Fuzzy Prolog as a natural application of CLP($\mathcal{R}$).

The rest of the paper is organized as follows. Section 2 describes the language and the 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 Language 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 logic predicate taking into account partial truth. Secondly we define aggegation operator to propagate truth value. Later we present the syntax and the different semantics of our fuzzy language, illustrating it with an example.

### 2.1 Truth value

Given a relevant universal set $X$, any arbitrary fuzzy set $A$ is defined by a function $A : X \to [0,1]$ unlike the crisp set that would be defined by a function $A : X \to \{0,1\}$. 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 the assignment of 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 $A : X \to \mathcal{E}([0,1])$, 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 definition 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 $A : X \to \mathcal{B}([0,1])$, where an element in $\mathcal{B}([0,1])$ is a countable union of sub-intervals of $[0,1]$.

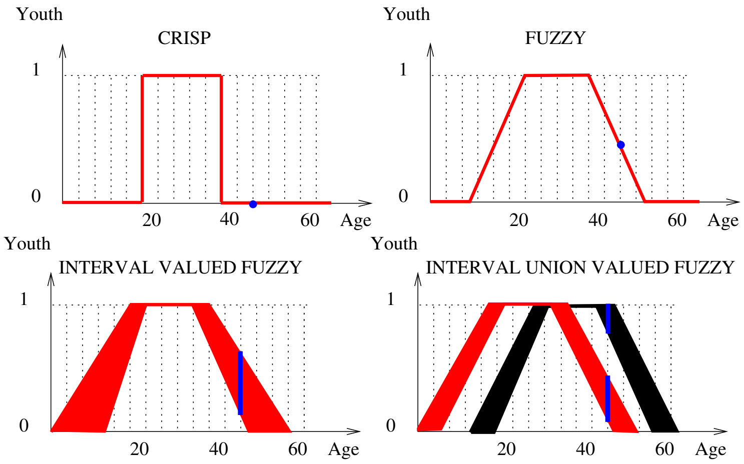



**Fig. 1.** Uncertainty level of a fuzzy predicate

These definitions of fuzzy sets entail different degrees of uncertainty. In Figure 1 we show the concept of *youth* with these different representations.

The level of uncertainty is increasing from the simple fuzzy function, where every age has only one real number representing its youth, to one where an interval represents, for example, the concept of youth of a group of people with slightly different definitions of the borders of the function. However, if we ask two different groups of people, for example people from two different continents, we might obtain a representation like the last one. The truth value of youth for 45 years has evolved from the value 0, to the value 0.5 in the simple fuzzy definition, later to the interval $[0.2, 0.5]$ and finally to the union of intervals $[0.2, 0.5] \bigcup [0.8, 1]$.

There are many usual situations that can only be represented by this general representation of truth value. Here we have two simple examples with their representation in our fuzzy language:

- Example 1: My sons are 16 and 18 years old. My neighbour's daughter, Jane, has the same age as one of my sons but I do not remember which one. If I consider the simple fuzzy definition of truth, then I can say that Jane is young with a truth value $V \in ([0.8] \bigcup [0.9]) \neq [0.85]$.

```
young(jane) :- [0.9] v [0.8].
```

- Example 2: New Laptop is a branch of computers with two laptop models (VZX and VZY). One model is very slow and the other one is very fast. If a client buys a New Laptop computer, the truth value of its speed will be $V \in ([0.02, 0.08] \bigcup [0.75, 0.90])$.

```
fail(newLaptop) :- [0.02, 0.08] v [0.75, 0.90].
```

Where each truth value is a union of intervals. The intervals in the first example represent the particular case of intervals consisting of only one element.

### 2.2 Aggregation Operators

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 done using the application of a numeric operator of the form $f : [0, 1]^n \to [0, 1]$. If it verifies $f(0, \dots, 0) = 0$ and $f(1, \dots, 1) = 1$, 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 generalizing *aggregation operator* of numbers to *aggregation operator* of intervals if we deal with the second definition of fuzzy sets. Following the theorem proven by Nguyen and Walker in [NW00] to extend T-norms and T-conorms to intervals, we propose the next definition:

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

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

Actually, we work with union of intervals and propose the definition:

**Definition 2 (union-aggregation).** *Given an interval-aggregation $F : \mathcal{E}([0,1])^n \to \mathcal{E}([0,1])$ defined over intervals, a union-aggregation $\mathcal{F} : \mathcal{B}([0,1])^n \to \mathcal{B}([0,1])$ is 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 *$\Sigma$-formula* where $\Sigma$ 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 $solvable(c)$.

### 2.3 Fuzzy 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* and we obtain information from the program through *fuzzy queries*. 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 $\mathcal{B}([0,1])$ expressed as constraints over the domain $[0,1]$.*

**Definition 4 (fuzzy clause).** *Let $A, B_1, \ldots, B_n$ be atoms,*

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

*is a* fuzzy clause *where $F$ is an interval-aggregation operator of truth values in $\mathcal{B}([0,1])$ represented as constraints over the domain $[0,1]$, where $F$ induces a union-aggregation as by definition 2.*

**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 in $\mathcal{B}([0,1])$.*

When we talk about constraints, we refer to expressions as: $(v \geq 0.5 \ \wedge \ v \leq 0.7) \ \vee \ (v \geq 0.8 \ \wedge \ v \leq 0.9)$ to represent a truth value in $[0.5, 0.7] \bigcup [0.8, 0.9]$, for example.

### 2.4 Least Model 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 $V_I$, to assign a truth value, in $\mathcal{B}([0,1])$, to each element of $B_I$.*

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

**Definition 7 (interval inclusion $\subseteq_{II}$).** *Given two intervals $I_1 = [a, b]$, $I_2 = [c, d]$ in $\mathcal{E}([0,1])$, $I_1 \subseteq_{II} I_2$ if and only if $c \leq a$ and $b \leq d$.*

**Definition 8 (Borel inclusion $\subseteq_{BI}$).** *Given two unions of intervals $U = I_1 \cup \ldots \cup I_N$, $U' = I'_1 \cup \ldots \cup I'_M$ in $\mathcal{B}([0,1])$, $U \subseteq_{BI} U'$ if and only if $\forall I_i \in U$ $\exists I'_j \in U'$ . $I_i \subseteq_{II} I'_j$ where $i \in 1..N$, $j \in 1..M$.*

**Definition 9 (interpretation inclusion $\sqsubseteq$).** *$I \sqsubseteq I'$ if and only if $B_I \subseteq B_{I'}$ and for all $B \in B_I$, $V_I(B) \subseteq_{II} V_{I'}(B)$, where $I = \langle B_I, V_I \rangle$, $I' = \langle B_{I'}, V_{I'} \rangle$ are interpretations.*

**Definition 10 (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 $v \subseteq_{II} V_I(A)$.*
- *$I$ 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 $v \subseteq_{II} V_I(A)$, where $\mathcal{F}$ is the union aggregation obtained from $F$.*
- *$I$ 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 $P$.

### 2.5 FixedPoint Semantics

The fixedpoint semantics we present is based on a one-step consequence operator $T_P$. The least fixedpoint $lfp(T_P) = I$ (i.e. $T_P(I) = I$) is the declarative meaning of the program $P$, so is equal to $lm(P)$.

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

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

$$B_{I'} = \{A \in B_P \mid Cond\}$$

$$V_{I'}(A) = \bigcup \{v \in \mathcal{B}([0,1]) \mid Cond\}$$

where

$$\begin{aligned} Cond = (&A \leftarrow v \text{ is a ground instance of a fact in } P \\ &\text{and } solvable(v)) \\ &\text{or} \\ (&A \leftarrow_F A_1, \ldots, A_n \text{ is a ground instance of a clause in } P, \\ &\{A_1, \ldots, A_n\} \subseteq B_I \\ &\text{and } solvable(v), v = \mathcal{F}(V_I(A_1), \ldots, V_I(A_n))). \end{aligned}$$

### 2.6 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, $\sigma$ is a substitution representing the instantiation of variables needed to get to this state from the initial one and $S$ is a constraint that represents the truth value of the goal at this state.

When computation starts, $A$ is the initial goal, $\sigma = \emptyset$ and $S$ is true (if there are neither previous instantiations nor initial constraints). When we get to 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 $P$, $\theta$ is the mgu of $a$ and $h$, and $\mu_a$ is the truth variable for $a$, and $solvable(S \wedge \mu_a = v)$.
- $\langle A \cup a, \sigma, S \rangle \rightarrow \langle (A \cup B)\theta, \sigma \cdot \theta, S \wedge c \rangle$
  if $h \leftarrow_F B$ is a rule of the program $P$, $\theta$ is the mgu of $a$ and $h$, $c$ is the constraint that represents the truth value obtained applying the aggregator $F$ on the truth variables of $B$, and $solvable(S \wedge c)$.
- $\langle A \cup a, \sigma, S \rangle \rightarrow fail$
  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 $B = \{p(\widehat{x})\sigma | \langle p(\widehat{x}), \emptyset, true \rangle \rightarrow^* \langle \emptyset, \sigma, S \rangle \}$ is the set of elements of the herbrand Base that are instantiated and that have succeeded;

and $V(p(\widehat{x})) = \cup \{v | \langle p(\widehat{x}), \emptyset, true \rangle \rightarrow^* \langle \emptyset, \sigma, S \rangle$, and $v$ is the solution of $S\}$ is the set of truth values of the elements of $B$ that is the union (got by backtraking) of truth values that are obtained from the set of constraints provided by the program P while query $p(\widehat{x})$ is computed.

Let's see an example. Suppose we have the following program:

$tall(john) \leftarrow 0.7$
$swift(john) \leftarrow [0.6, 0.8]$
$good\_player(X) \leftarrow_{luka} tall(X), swift(X)$

Here, we have two facts, $tall(john)$ and $swift(john)$ whose truth values are the unitary interval $[0.7, 0.7]$ and the interval $[0.6, 0.8]$, respectively, and a clause for the *good_player* predicate whose *aggregation operator* is the Lukasiewicz T-norm.

Consider the fuzzy goal

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

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:

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

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

$$\langle \{\}, \{X = john\}, \mu = max(0, \mu_{tall} + \mu_{swift} - 1) \wedge \mu_{tall} = 0.7 \wedge 0.6 \leq \mu_{swift} \wedge \mu_{swift} \leq 0.8 \rangle$$

As $\mu = max(0, \mu_{tall} + \mu_{swift} - 1) \wedge \mu_{tall} = 0.7 \wedge 0.6 \leq \mu_{swift} \wedge \mu_{swift} \leq 0.8$ entails $\mu \in [0.3, 0.5]$, the answer to the query $good\_player(X)$ is $X = john$ with truth value in the interval $[0.3, 0.5]$.

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

## 3 Implementation and Syntax

### 3.1 CLP($\mathcal{R}$)

Constraint Logic Programming [JL87] began as a natural merging 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.

We decided to implement this interpreter as a syntactic extension of a CLP($\mathcal{R}$) system. CLP($\mathcal{R}$) was incorporated as a library in the Ciao Prolog system[1].

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 permit having 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.

### 3.2 Syntax

Each fuzzy Prolog clause has an additional argument in the head which represents its truth value in terms of the truth values of the subgoals of the body of the clause. A fact $A \leftarrow v$ is represented by a Fuzzy Prolog fact that describes the range of values of $v$ with a union of intervals (that can be only an interval or even a real number in particular cases). The following examples illustrate the concrete syntax of programs:

$$
\begin{array}{l}
youth(45) \leftarrow \\
\quad [0.2, 0.5] \bigcup [0.8, 1] \\
tall(john) \leftarrow 0.7 \\
swift(john) \leftarrow \\
\quad [0.6, 0.8] \\
good\_player(X) \leftarrow_{min} \\
\quad tall(X), \\
\quad swift(X)
\end{array}
$$

```
youth(45) :~
     [0.2,0.5] v [0.8,1].
tall(john) :~ 0.7.
tall(john) :~
     [0.6,0.8].
good_player(X) :~ min
     tall(X),
     swift(X).
```

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

These clauses are expanded at compilation time to constrained clauses that are managed by CLP($\mathcal{R}$) at run-time. Predicates $. = ./2$, $. < ./2$, $. <= ./2$, $. > ./2$ and $. >= ./2$ are the Ciao CLP($\mathcal{R}$) operators for representing constraint inequalities. For example the first fuzzy fact is expanded to these Prolog clauses with constraints

```
youth(45,V):-   V .>=. 0.2,
                V .<.  0.5.
youth(45,V):-   V .>=. 0.8,
                V .<.  1.
```

And the fuzzy clause

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

is expanded to

```
p(X,Vp) :- q(X,Vq),r(X,Vr),
           minim([Vq,Vr],Vp),
           Vp .>=. 0, Vp .=<. 1.
```

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.

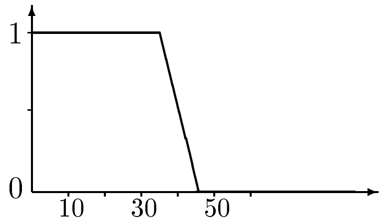
### 3.3 Syntax Sugar

Fuzzy predicates with piecewise linear continuous membership functions like young/2 in Figure 2 can be written in a compact way:

```
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 so prefer. The code expansion is the following:

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




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

**Fig. 2.** Fuzzy predicate young/2

```
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 fails and 1 otherwise.

We also provide 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,V) :-
        p_f(X,Y,Vp),
        V .=. 1 - Vp.
```

### 3.4 Example

A simple example could be trying to measure 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. We deal with the fuzzy concepts *small* and *large* (Figure 3):

```
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`. With our fuzzy Prolog we can use any of them as, in the following two programs:

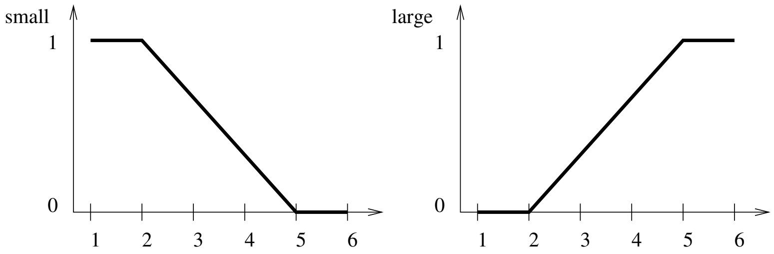



**Fig. 3.** Fuzzy predicates small/2 and large/2

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

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

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

sum(5) :~ max
      two_dice(4,1),
      two_dice(1,4),
      two_dice(3,2),
      two_dice(2,3).

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

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

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

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

sum(5) :~ dprod
      two_dice(4,1),
      two_dice(1,4),
      two_dice(3,2),
      two_dice(2,3).

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

`two_dice(X,Y)` represents the possibility that the first die gives `X` and at the same time the second die gives `Y`. The predicate `sum(5)` aggregates the possibilities of the four cases in which the two dice can sum 5. To consult the truth value of a goal we are going to use an additional argument, i.e. `sum(5,V)`. Other syntax is to use another predicate `truth(Goal,V)` to obtain the truth value $V$ of a goal *Goal*. In this case, it is equivalent to `truth(sum(5),V)`. In the consults of our example we can observe the different provided answers for each aggregation operator.

## 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. This extension to Prolog is realized by 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 considers only one operator to get the truth value of the fuzzy clauses. We have generalized all operators through the concept of aggregation and this makes our Fuzzy Prolog subsume all the means 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.

Presently we are working in several related issues:

- Solving any semantic problems that arise when we try to combine crisp and fuzzy logic in the same programming language.
- Obtaining constructive answers to any kind of goal.
- Implementing the expansion over other CLP($\mathcal{R}$) systems.

## Acknowledgement

The authors would like to thank the suggestions of Enric Trillas and Francisco Bueno in improving the content and the ideas behind the paper. This work is partly supported by the Spanish project EDIPIA [MCYT TIC 99-1151].

## References

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

[BMR01] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint logic programming: syntax and semantics. In *ACM TOPLAS*, volume 23, pages 1–29, 2001.

[CH00] D. Cabeza and M. Hermenegildo. A New Module System for Prolog. In *International Conference on Computational Logic, CL2000*, number 1861 in LNAI, pages 131–148. Springer-Verlag, July 2000.

[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 9*, volume 2, 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(r) 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.

[KMP] E.P. Klement, R. Mesiar, and E. Pap. Triangular norms. Kluwer Academic Publishers.

[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, Boca Raton, 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.

[Sha83] Ehud Y. Shapiro. Logic programs with uncertainties: A tool for implementing rule-based systems. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 529–532, Karlsruhe, West Germany, August 1983.

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