Fuzzy Prolog: A new approach using soft constraints propagation

S. Guadarrama [a], C. Vaucheret [a], S. Muñoz [b]

[a]Department of Artificial Intelligence,
Universidad Politécnica de Madrid,
Campus de Montegancedo, 28660 Madrid, Spain

[b]Department of Languages and Systems of the Information
Universidad Politécnica de Madrid,
Campus de Montegancedo, 28660 Madrid, Spain

We present a definition of a Fuzzy Prolog Language that models $\mathcal{B}([0,1])$-valued Fuzzy Logic, and subsumes former approaches because it uses a truth value representation based on a union of sub-intervals on [0,1] and it is defined using general operators that can model different logics. This extension to Prolog is realized by interpreting fuzzy reasoning as a set of constraints which are propagated through the rules by means of aggregation operators. It is given the declarative and procedural semantics for Fuzzy Logic programs and it is proven their equivalence. In addition, we present the implementation of an interpreter for this conceived language using Constraint Logic Programming over Real numbers (CLP($\mathcal{R}$)).

Keywords

Fuzzy Prolog, Modeling Uncertainty, Fuzzy 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 [2], examples being the Prolog-Elf system [3], Fril Prolog system [4] and the F-Prolog language [5]. However, there was no common method for fuzzifying Prolog as it has been noted in [6]. 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 [7].

Corresponding authors: email@adress ,email@adress
A preliminary version of this paper appeared as [1] This research was supported in part by ...

On the other hand in [8] logic programming is considered a useful tool for implementing methods for reasoning with uncertainty.

There is also an extension of constraint logic programming [9], 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.

Recently, it has been appeared a theoretical model for fuzzy logic programming without negation [10], which deals with many values implications.

During last years it has been published several papers ([11–13]) about multi-adjoint programming, where it is described a theoretical model for it but where there is not presented a way to implemented it.

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

1. A truth value will be a finite union of subintervals 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 [14] as min, prod, etc), disjunctive operators [15](triangular co-norms as max, sum, etc), average operators (averages as arithmetic average, quasi-linear average, etc) and hybrid operators (combinations of previous operators [16]).

3. Crisp and fuzzy reasoning are combined into a Prolog compiler consistently.

4. It is given the declarative and procedural semantics for Fuzzy Logic programs and it is proven their equivalence.

5. It is presented a implementation of the proposed language.

We add fuzzyness 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 built-in 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 [17], an interpretation of truth values as intervals, but we are proposing 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 fuzzy and 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($\mathcal{R}$). Section 4 discuss about Close Word Assumption. Section 5 explain how can be combined crisp and fuzzy logic in a consistent way. Section 6 show two examples using this language and their results. Finally, we conclude and discuss some future work (Section 7).

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 aggregation operators to propagate truth value. Later we present the syntax and the different semantics of our fuzzy language, illustrating it with an example, and it is proved the equivalence between them.

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* [18,17]. 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

$A : X \rightarrow \mathcal{B}([0,1])$, where an element in $\mathcal{B}([0,1])$ is a countable union of sub-intervals of $[0,1]$.

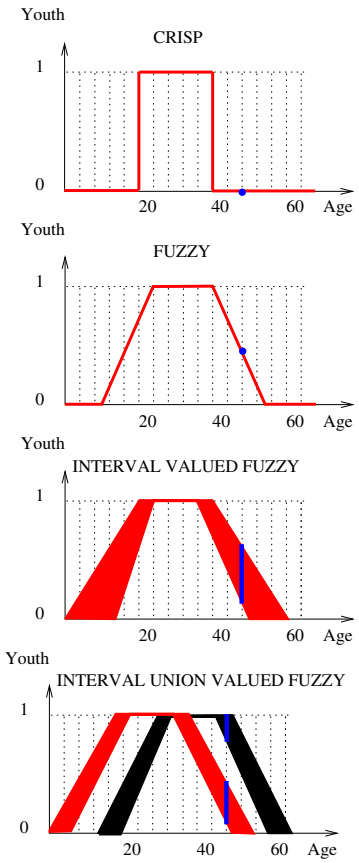


Figure 1. Definition level of a fuzzy predicate

These definitions of fuzzy sets entail different degrees of fuzzyness. In Figure 1 we show the concept of *youth* with four different interpretations.

The level of fuzzyness is increasing from the crisp function or 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 in the crisp model, 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 father is 45 years old. If someone ask me about his degree of young I would assign $V \in [0.2, 0.5]$, but if someone ask him about his degree of young he would assign himself $V \in [0.8, 1]$. And we can say that he is young with $V \in ([0.2, 0.5] \bigcup [0.8, 1])$.

- Example 2: 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 $V \in ([0.1,0.7] \bigcup [0.5,0.9]) \neq [0.3,0.8]$.

```
age(jane,16).
age(jane,18).
youth(16) :- [0.1,0.7].
youth(17) :- [0.3,0.8].
youth(18):- [0.4,0.9].
young(jane) :- age(jane,A), youth(A).
```

- Example 3: New Laptop is a branch of computers with a laptop models. This model is very slow when works with graphic applications but it is very fast when works with office applications. If a client buys a New Laptop computer, the truth value of its speed will be $V \in ([0.02, 0.08] \bigcup [0.75, 0.90])$, but depending of its use, its speed could the lowest the highest or even an average.

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

Where each truth value is a union of intervals and can be used for this three different examples.

2.2. Fuzzyness versus Uncertainty

Thanks to this powerful notation we can model many real problems but it is interesting to notice that with this truth value representation we can handle both uncertainty and fuzzyness at the same time.

We are going to come back to the example 2 of the above subsection. We will represent the truth value of the concept of youth with an interval as in the fourth representation in the Figure 1, instead of using real numbers. We can say that, in that case, Jane is young with a truth value $V \in$ ([0.1,0.7] $\bigcup$ [0.4,0,9]). It is a union of intervals which is representing uncertainty because we do not know which of the two intervals represents the youth of Jane (we do not know which one of the two is her age).

In the example 1 presented before shows the truth value of youth of 45 years old that is $[0.1, 0.4] \bigcup [0.8, 1]$. It is a union of intervals which, in this case, is representing fuzzyness because the concept of youth is represented with the maximum level of fuzzyness. We know that the age is 45 and the truth value that represents its youth is fuzzy.

Although both representations are semantically different, they are tackled using the same syntax in a sound way as we will describe below.

2.3. Aggregation Operators

The truth value of a goal will depend on the truth value of the subgoals which are in the body of the clauses of its definition. We use *aggregation operators* [19] 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 $f : [0,1]^n \rightarrow [0,1]$. 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 and Prade, in [20], propose 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*).

If we deal with the second definition of fuzzy sets it is necessary to generalize from *aggregation operators* of numbers to *aggregation operators* of intervals. Following the theorem proven by Nguyen and Walker in [17] to extend T-norms and T-conorms to intervals, we propose the next definitions.

Definition 2.1 (interval-aggregation) *Given an aggregation* $f : [0,1]^n \rightarrow [0,1]$, *an interval-aggregation* $F : \mathcal{E}([0,1])^n \rightarrow \mathcal{E}([0,1])$ *is defined as follows:*

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

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

Definition 2.2 (union-aggregation) *Given an interval-aggregation* $F : \mathcal{E}([0,1])^n \rightarrow \mathcal{E}([0,1])$ *defined over intervals, a union-aggregation* $\mathcal{F} : \mathcal{B}([0,1])^n \rightarrow \mathcal{B}([0,1])$ *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 [21], 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 $solvable(c)$.

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

2.4. 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 2.3 (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 2.4 (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, which induces a union-aggregation, as by definition 2.2, $\mathcal{F}$ of truth values in $\mathcal{B}([0,1])$ represented as constraints over the domain $[0,1]$.*

Definition 2.5 (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 $\mathcal{B}([0,1])$.

2.5. Least Model Semantics

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

Definition 2.6 (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 2.7 (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 2.8 (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 2.9 (interpretation inclusion $\sqsubseteq$) *$I \sqsubseteq I'$ if and only if $B_I \subseteq B_{I'}$ and for all $B \in B_I$, $V_I(B) \subseteq_{BI} V_{I'}(B)$, where $I = \langle B_I, V_I \rangle$, $I' = \langle B_{I'}, V_{I'} \rangle$ are interpretations.*

Definition 2.10 (valuation) *A* valuation σ *of an atom A is an assignment of elements of U to variables of A. So $\sigma(A) \in B$ is a ground atom.*

Definition 2.11 (model) *Given an* interpretation $I = \langle B_I, V_I \rangle$

- *I is a* model *for a* fuzzy fact $A \leftarrow v$, *if for all valuation σ, $\sigma(A) \in B_I$ and $v \subseteq_{BI} V_I(\sigma(A))$.*

- *I is a* model *for a clause* $A \leftarrow_F B_1, \ldots, B_n$ *when the following holds: for all valuation* σ*, if* $\sigma(B_i) \in B_I$*,* $1 \leq i \leq n$*, and* $v = \mathcal{F}(V_I(\sigma(B_1)), \ldots, V_I(\sigma(B_n)))$ *then* $\sigma(A) \in B_I$ *and* $v \subseteq_{BI} V_I(\sigma(A))$*, 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.*

Every program has a least model which is usually regarded as the intended interpretation of the program since it is the most conservative model. Let $\cap$ be the meet operator on the lattice of interpretations $(I, \sqsubseteq)$, then we can prove the following result.

Theorema 2.1 (model intersection property) *Let* $I_1 = \langle B_{I_1}, V_{I_1} \rangle$,$I_2 = \langle B_{I_1}, V_{I_1} \rangle$ *be models of a fuzzy program* P*. Then* $I_1 \cap I_2$ *is a model of* P*.*

Proof. Let $M = \langle B_M, V_M \rangle = I_1 \cap I_2$. Since I_1 and I_2 are models of P, they are models for each fact and clause of P. Then for all valuation σ we have

- for all fact $A \leftarrow v$ in P,
  - $\sigma(A) \subseteq B_{I_1}$ and $\sigma(A) \in B_{I_2}$ then $\sigma(A) \in B_{I_1} \cap B_{I_2} = B_M$,
  - $v \subseteq_{BI} V_{I_1}(\sigma(A))$ and $v \subseteq_{BI} V_{I_2}(\sigma(A))$, then $v \subseteq_{BI} V_{I_1}(\sigma(A)) \cap V_{I_2}(\sigma(A)) = V_M(\sigma(A))$

  therefore M is a model for $A \leftarrow v$
- and for all clause $A \leftarrow_F B_1, \ldots, B_n$ in P
  - if $\sigma(B_i) \in B_M$, $\sigma(B_i) \in B_{I_1}$ and $\sigma(B_i) \in B_{I_2}$. Then $\sigma(A) \in B_{I_1}$ and $\sigma(A) \in B_{I_2}$. Hence $\sigma(A) \in B_{I_1} \cap B_{I_2} = B_M$.
  - if $v = \mathcal{F}(V_M(\sigma(B_1)), \ldots, V_M(\sigma(B_n)))$, since F is monotonic, $v \subseteq_{BI} V_{I_1}(\sigma(A))$ and $v \subseteq_{BI} V_{I_2}(\sigma(A))$, then $v \subseteq_{BI} V_{I_1}(\sigma(A)) \cap V_{I_2}(\sigma(A)) = V_M(\sigma(A))$

  therefore M is a model for $A \leftarrow_F B_1, \ldots, B_n$

and M is model of P.

Remark 2.1 (Least model semantic) *If we let* $\mathbf{M}$ *be the set of all models of a program* P*, the intersection of all of this models,* $\bigcap \mathbf{M}$*, is a model and it is the least model of* P*. We denote the least model of a program* P *by* $lm(P)$*.*

Example 2.1 *Let's see an example. Suppose we have the following program* P*:*

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

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.*

The following interpretation $I = \langle B, V \rangle$ *is a model for* P*, where*

$B = \{tall(john), tall(peter), swift(john), good\_player(john), good\_player(peter)\}$ *and*

$$\begin{aligned} V(tall(john)) &= [0.7, 1] \\ V(swift(john)) &= [0.5, 0.8] \\ V(tall(peter)) &= [0.6, 0.7] \vee [0.8, 0.8] \\ V(good\_player(john)) &= [0.2, 0.9] \\ V(good\_player(peter)) &= [0.5, 0.9] \end{aligned}$$

note that for instance if $V(good\_player(john)) = [0.2, 0.5]$ $I = \langle B, V \rangle$ *cannot be a model of* P*, the reason is that* $v = luka([0.7, 1], [0.5, 0.8]) = [0.7 + 0.5 - 1, 1 + 0.8 - 1] = [0.2, 0.8] \not\subseteq_{II} [0.2, 0.5]$.

The least model of P *is the intersection of all models of* P *which is* $M = \langle B_M, V_M \rangle$ *where* $B_M = \{tall(john), tall(peter), swift(john), good\_player(john)\}$ *and*

$$\begin{aligned} V_M(tall(john)) &= [0.7, 0.7] \\ V_M(swift(john)) &= [0.6, 0.8] \\ V_M(tall(peter)) &= [0.6, 0.7] \vee [0.8, 0.8] \\ V_M(good\_player(john)) &= [0.3, 0.5] \end{aligned}$$

2.6. Fixed-Point Semantics

The fixed-point semantics we present is based on a one-step consequence operator T_P. The least

fixed-point $lfp(T_P) = I$ (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 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 $T_P(I) = I'$, $I' = \langle B_{I'}, V_{I'} \rangle$

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

where

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

The set of interpretations forms a complete lattice so that, T_P it is continuous.

Recall the definition of the *ordinal powers* of a function G over a complete lattice X:

$$G \uparrow \alpha = \begin{cases} \bigsqcup\{G \uparrow \alpha' \mid \alpha' < \alpha\} \\ \qquad \text{if } \alpha \text{ is a limit ordinal,} \\ G(G \uparrow (\alpha - 1)) \\ \qquad \text{if } \alpha \text{ is a successor ordinal,} \end{cases}$$

and dually,

$$G \downarrow \alpha = \begin{cases} \bigsqcap\{G \downarrow \alpha' \mid \alpha' < \alpha\} \\ \qquad \text{if } \alpha \text{ is a limit ordinal,} \\ G(G \downarrow (\alpha - 1)) \\ \qquad \text{if } \alpha \text{ is a successor ordinal,} \end{cases}$$

Since the first limit ordinal is 0, it follows that in particular, $G \uparrow 0 = \bot_X$ (the bottom element of the lattice X) and $G \downarrow 0 = \top_X$ (the top element). From Kleene's fixed point theorem we know that the least fixed-point of any continuous operator is reached at the first infinite ordinal ω. Hence $lfp(T_P) = T_P \uparrow \omega$.

Example 2.2 *Consider the same program P of the example 2.1, the ordinal powers of T_P are*

$T_P \uparrow 0 = \{\}$
$T_P \uparrow 1 = \{tall(john), swift(john),$
$tall(peter)\}$ *and*

$$\begin{array}{lcl} V(tall(john)) & = & [0.7, 0.7] \\ V(swift(john)) & = & [0.6, 0.8] \\ V(tall(peter)) & = & [0.6, 0.7] \vee [0.8, 0.8] \end{array}$$

$T_P \uparrow 2 = \{tall(john), swift(john), tall(peter),$
$good\_player(john)\}$ *and*

$$\begin{array}{lcl} V(tall(john)) & = & [0.7, 0.7] \\ V(swift(john)) & = & [0.6, 0.8] \\ V(tall(peter)) & = & [0.6, 0.7] \vee [0.8, 0.8] \\ V(good\_player(john)) & = & [0.3, 0.5] \end{array}$$

$T_P \uparrow 3 = T_P \uparrow 2$.

Lemma 2.1 *Let P a fuzzy program, M is a model of P if and only if M is a prefixpoint of T_P, that is $T_P(M) \sqsubseteq M$.*

Proof. Let $M = \langle B_M, V_M \rangle$ and $T_P(M) = \langle B_{T_P}, V_{T_P} \rangle$.

We first prove the "if" direction. Let A be an element of B_{T_P}, then by definition of T_P there exists a ground instance of a fact of P, $A \leftarrow v$, or a ground instance of a clause of P, $A \leftarrow_F A_1, \ldots, A_n$ where $\{A_1, \ldots, A_n\} \subseteq B_M$ and $v = \mathcal{F}(V_M(A_1), \ldots, V_M(A_n))$. Since M is a model of P, $A \in B_M$, and each $v \subseteq_{BI} V_M(A)$, then $V_{T_P}(A) \subseteq_{BI} V_M(A)$ and then $T_P(M) \sqsubseteq M$. □

Analogously, for the "only if" direction, for each ground instance where $\{A_1, \ldots, A_n\} \subseteq B_M$, $v = \mathcal{F}(V_M(A_1), \ldots, V_M(A_n))$, $A \in B_{T_P}$ and $v \subseteq_{BI} V_{T_P}(A)$, but as $T_P(M) \subseteq M$, $B_{T_P} \subseteq B_M$ and $V_{T_P}(A) \subseteq_{BI} V_M(A)$. Then $A \in B_M$ and $v \subseteq_{BI} V_M(A)$ therefore M is a model of P. □

Given this relationship, it is straightforward to prove that the least model of a program P is also the least fixed-point of T_P.

Theorema 2.2 *Let P be a fuzzy program, $lm(P) = lfp(T_P)$.*

Proof.

$$\begin{aligned} lm(P) &= \bigcap\{M \mid M \text{ is a model of } P\} \\ &= \bigcap\{M \mid M \text{ is a pre-fixpoint of } P\} \\ &\quad \text{from lemma 2.1} \\ &= lfp(T_P) \quad \text{by the Knaster-Tarski} \\ &\quad \text{Fixpoint Theorem [22]}\Box \end{aligned}$$

2.7. Operational Semantics

The procedural semantics is interpreted as a sequence of transitions between different states of a 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 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:

1. $\langle A \cup a, \sigma, S\rangle \rightarrow \langle A\theta, \sigma \cdot \theta, S \wedge \mu_a = v\rangle$
   if $h \leftarrow v$ is a fact of the program P, θ is the mgu of a and h, μ_a is the truth value for a and $solvable(S \wedge \mu_a = v)$.
2. $\langle A \cup a, \sigma, S\rangle \rightarrow \langle (A \cup B)\theta, \sigma \cdot \theta, S \wedge c\rangle$
   if $h \leftarrow_F B$ is a rule of the program P, θ is the mgu of a and h, c is the constraint that represents the truth value obtained applying the union-aggregation $\mathcal{F}$ to the truth values of B, and $solvable(S \wedge c)$.
3. $\langle A \cup a, \sigma, S\rangle \rightarrow fail$
   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 backtracking) of truth values that are obtained from the set of constraints provided by the program P while query $p(\widehat{x})$ is computed.

Example 2.3 *Let P be the program of example 2.1. Consider the fuzzy goal*

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

the first transition *in the computation is*

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

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 the interval $[0.3, 0.5]$.

In order to prove the equivalence between operational semantic and fixed-point semantic, it is useful to introduce a type of canonical top-down evaluation strategy. In this strategy all literals are reduced at each step in a derivation. For obvious reasons, such a derivation is called *breadth-first.*

Definition 2.12 (Breadth-first transition)

Given the following set of valid transitions:

$$\begin{array}{l}\langle\{\{A_1,\ldots,A_n\},\sigma,S\rangle \rightarrow \\ \qquad \langle\{\{A_2,\ldots,A_n\}\cup B_1,\sigma\cdot\theta_1,S\wedge c_1\rangle \\ \langle\{\{A_1,\ldots,A_n\},\sigma,S\rangle \rightarrow \\ \qquad \langle\{\{A_1,A_3\ldots,A_n\}\cup B_2,\sigma\cdot\theta_2,S\wedge c_2\rangle \\ \qquad\qquad \vdots \\ \langle\{\{A_1,\ldots,A_n\},\sigma,S\rangle \rightarrow \\ \qquad \langle\{\{A_1,\ldots,A_{n-1}\}\cup B_n,\sigma\cdot\theta_n,S\wedge c_n\rangle\end{array}$$

a breadth-first transition *is defined as*
$\langle\{A_1,\ldots,A_n\},\sigma,S\rangle \rightarrow_{BF}$
$\langle B_1\cup\ldots\cup B_n,\sigma\cdot\theta_1\cdot\ldots\cdot\theta_n,S\wedge c_1\wedge\ldots\wedge c_n\rangle$
in which all literals are reduced at one step.

For example a *breadth-first derivation* of the above example is:

$$\begin{array}{l}\langle\{(good\_player(X)\},\epsilon,true\rangle \rightarrow_{BF} \\ \langle\{tall(X),swift(X)\},\epsilon, \\ \mu = max(0,\mu_{tall}+\mu_{swift}-1)\rangle \rightarrow_{BF}\end{array}$$

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

Theorem 2.3 *Given a ordinal number n and $T_P\uparrow n=\langle B_{T_{P_n}},V_{T_{P_n}}\rangle$. there is a successful breadth-first derivation of lengh less or equal to $n+1$ for a program P, $\langle\{A_1,\ldots,A_k\},\sigma,S_1\rangle\rightarrow^*_{BF}$ $\langle\emptyset,\theta,S_2\rangle$ iff $A_i\theta\in B_{T_{P_n}}$ and $solvable(S\wedge\mu_{A_i}=v_i)$ and $v_i\subseteq_{BI} V_{T_{P_n}}(A_i\theta)$.*

Proof. The proof is by induction on n. For the base case, all the literals are reduced using the first type of transitions, that is, for each literal A_i, it exits a fact $h_i\leftarrow v_i$ such that θ_i is the mgu of A_i and h_i, and μ_{A_i} is the truth variable for A_i, and $solvable(S_1\wedge\mu_{A_i}=v_i)$. By definition of T_P, each $A_i\theta_i\in B_{T_{P_1}}$ and each $v_i\subseteq_{BI} V_{T_{P_1}}(A_i\theta)$ where $\langle B_{T_{P_1}},V_{T_{P_1}}\rangle=T_P\uparrow 1$.

For the general case, consider the successful derivation,
$\langle\{A_1,\ldots,A_k\},\sigma_1,S_1\rangle\rightarrow_{BF}\langle B,\sigma_2,S_2\rangle\rightarrow_{BF}\ldots$
$\ldots\rightarrow_{BF}\langle\emptyset,\sigma_n,S_n\rangle$
consider the transition

$$\langle\{A_1,\ldots,A_k\},\sigma_1,S_1\rangle\rightarrow_{BF}\langle B,\sigma_2,S_2\rangle$$

When a literal A_i is reduced using a fact the result is the same as in the base case, otherwise there is a clause $h_i\leftarrow_F B_{1_i},\ldots,B_{m_i}$ in P such that θ_i is the mgu of A_i and $h_i\in B\sigma_2$ and $B_{j_i}\theta_i\in B\sigma_2$, by the induction hypothesis $B\sigma_2\subseteq B_{T_{P_{n-1}}}$ and $solvable(S_2\wedge\mu_{B_{j_i}}=v_{j_i})$ and $v_{j_i}\subseteq_{BI} V_{T_{P_{n-1}}}(B_{j_i}\sigma_2)$ then $B_{j_i}\theta_i\subseteq B_{T_{P_{n-1}}}$ and by definition of T_P, $A_i\theta_i\in B_{T_{P_n}}$ and $solvable(S_1\wedge\mu_{A_i}=v_i)$ and $v_i=\subseteq_{BI} V_{T_{P_n}}(A_i\sigma_1)$. □

Theorem 2.4 *For a program P there is a successful derivation*

$$\langle p(\widehat{x}),\emptyset,true\rangle\rightarrow^*\langle\emptyset,\sigma,S\rangle$$

iff $p(\widehat{x})\sigma\in B$ and v is the solution of S and $v\subseteq_{BI} V(p(\widehat{x})\sigma)$ where $lfp(T_P)=\langle B,V\rangle$

Proof. It follows from the fact that $lfp(T_P)=T_P\uparrow\omega$ and from the Theorem 2.3. □

Theorem 2.5 *For a fuzzy program P the three semantics are equivalent, i.e.*

$$SS(P)=lfp(TP)=lm(P)$$

Proof. the first equivalence follows from Theorem 2.4 and the second from Theorem 2.2. □

3. Implementation and Syntax

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

Constraint Logic Programming [23] 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($\mathcal{R}$) [24] 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

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

been designed with modular incremental compilation in mind. Its module system [25] 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:

$youth(45) \leftarrow$	`youth(45):~`
$[0.2, 0.5] \bigcup [0.8, 1]$	`[0.2,0.5]v[0.8,1].`
$tall(john) \leftarrow 0.7$	`tall(john):~ 0.7.`
$swift(john) \leftarrow$	`swift(john):~`
$[0.6, 0.8]$	`[0.6,0.8].`
$good\_player(X) \leftarrow_{min}$	`good_player(X):~min`
$tall(X)$,	`tall(X),`
$swift(X)$	`swift(X).`

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

```
      good_player(X) :~ min
   tall(X), swift(X).
```

is expanded to

```
good_player(X,Vp) :- tall(X,Vq),swift(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. and in a similar way 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. Fuzzy Negation

We also provide the possibility of defining a predicate that is the fuzzy negation of a given fuzzy predicate. For example, $not\_young/2$ can be defined from $young/2$ (see figure 2) with the following line:

```
not_young :# fnot  young/2.
```

that is expanded at compilation time as:

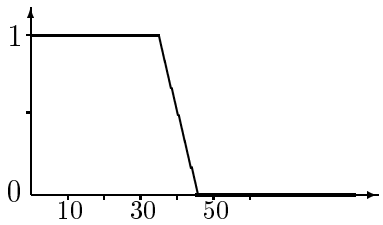
```
not_young(X,V) :-
        young(X,Vp),
        V .=. 1 - Vp.
```

3.4. Syntactic 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:




```
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.
```

Figure 2. Fuzzy predicate young/2

4. Closed World Assumption

We have to set a semantics to interpret our fuzzy predicates. We have thought about two main possibilities. E.g., if we have the definition of *age_about_*21/2 as

```
age_about_21(john,1):~ . age_about_21(susan,0.7):~
```

where the goal *age_about_*21(X,V) success with $X = john$ and $V = 1$ or with $X = susan$ and $V = 0.7$. If we want to work with the Closed World Assumption (CWA) [26] then we will obtain $V = 0$ for any other value of X different from *john* and *susan*. The meaning is that the predicate is defined for all values and the membership value will be 0 if the predicate is not explicitly defined with other value. In this example we know that the age of *john* and *susan* is about 21 and with CWA we are saying too that the rest of the people are not about 21. This is the equivalent semantics to the one in crisp definitions but we think that we usually prefer to mean something different, i.e. in this case we can mean that we know that *john* and *susan* are about 21 and that we have no information about the age of the rest of people. Therefore we do not know if the age of *peter* is about 21 or not; and if we know that *nick*'s age is not about 21 we can explicitly declare

```
age_about_21(nick,0):~ .
```

We are going to work with this semantics for fuzzy predicates because we think it is the most alike to human reasoning. So a fuzzy goal can be true (value 1), false (value 0) or having other membership value. We have added the concept of `unknown` to represent no explicit knowledge in fuzzy definitions. The way of represent this new state is very simple, using the common failure of Prolog. We give it the meaning of `unknown` considering that the meaning that it has in crisp logic is not necessary here because in fuzzy logic is represented with truth value 0. E.g. with our definition of *age_about_*21/2 we will get

```
?- age_about_21(john,V).
    V = 1 ? ;
?- age_about_21(nick,V).
    V = 0 ? ;
?- age_about_21(peter,V).
    no
```

This means *john*'s age is about 21, *nick*'s age is not about 21 and we have no data about *peter*'s age.

We are going to use crisp Prolog predicates too and Prolog works with the CWA. This means that all information that is not explicitly true then is false. E.g., if we have the predicate definition of *student*/1 as

```
student(john). student(peter).
```

then we have that the goal *student*(X) successes with $X = john$ or with $X = peter$ but fails with any other value different from these. I.e, here we can have

```
?- student(john).
    yes
?- student(nick).
    no
```

that means that *john* is a student and *nick* is not. This is the semantics of Prolog and it is the one we are going to adopt for crisp logic because we want our system to be compatible with conventional Prolog reasoning. Notice the different interpretation of the Prolog failure for crisp or fuzzy goals.

5. Combining Crisp and Fuzzy Logic

Sometimes we use definitions of fuzzy predicates that include crisp predicate calls (see work [27] for more details). The definition usually uses crisp predicate calls as requirements that data have to satisfy to verify the definition in a level superior to 0. E. g. if we can say that a mature student is a student whose age is about 21 then we can define the fuzzy predicate *mature_student*/2 as

```
mature_student(X):~
        student(X),
        age_about_21(X).
```

The way of interpret this definition is salty complicated if we realize there are subgoals in the body of the clause with two semantics because crisp and fuzzy predicates has different semantics as we have seen in 4. In this example what we want is that the goal $mature\_student(X, V)$ gives us $V = 0$ if X were not a student for a value of X, and that V had the corresponding value if X is an student and we know if his age is about 21, and that if X were a student but we do not know anything about his age then the result will be *unknown*. E.g. we can interpret

```
?- mature_student(john,V).
    .=.(V,1.0) ?
?- mature_student(susan,V).
    .=.(V,0.0) ?
?- mature_student(peter,V).
    no
```

as *john* is a mature student (he is a student and his age is about 21), *susan* is not a mature student (since she is not a student) and we do not know the value of maturity of *peter* as student because although he is a student, we do not know if his age is about 21.

5.1. Solution: Fuzzify crisp predicates

It is important to centre the problem to resolve. We realize that the good way to focus the problem of combining crisp and fuzzy logic is transforming crisp in fuzzy predicates before the combination and forthcoming we can deal with the more located problem of how to fuzzify a crisp predicate.

For the below example of crisp predicate *student*/1, we will produce the predicate *f_student*/2 that will be an equivalent fuzzy predicate to the crisp one.

```
f_student :# fuzzy student/1.
```

So there would be a new definition of *mature_student*/2 that will be

```
mature_student(X,M):~
        f_student(X,M1),
        age_about_21(X,M2).
```

For this reason the internal running of our fuzzy is very homogeneous because we only consider fuzzy subgoals in the body of the clause. Now the only and not simple problem is how to fuzzify crisp predicates.

5.2. Fuzzified Predicates

If we make a crisp predicate turns into fuzzy, then we have to keep the same semantics interpretation. For our example we could think in a first glance that the corresponding fuzzified definition will be

```
f_student(john, 1):~ . f_student(peter, 1):~ .
```

Nevertheless it is not semantically equivalent to *student*/1. With the crisp one we show that *john* and *peter* are students and that any other value of X in $student(X)$ would make the goal to fail. E.g. the goals

```
?- student(peter).
    yes
?- student(susan).
    no
```

means that *peter* is a student and *susan* not. However, the definition of *f_student* below means that *john* and *peter* are students and that we do not know which values of owing will be V for the

rest values of X in the goal $f\_student(X, V)$. I.e., the same goals

```
?- f_student(peter,V).
    .=.(V,1.0) ?
?- f_student(susan,V).
    no
```

means that *peter* is a student and that we do not know if *susan* is a student or not when what we would like to say is that we know that she is not a student. So, the answer that is semantically equivalent to the first ones will be

```
?- f_student(peter,V).
    .=.(V,1.0) ?
?- f_student(susan,V).
    .=.(V,0.0) ?
```

This is the way of work of our fuzzified predicates, it is the only sound transformation that keep the same meaning than in the crisp predicates. We have tried to get an automatic method to obtain the equivalent fuzzy predicate to a crisp one keeping this semantics.

The first simple approach is to use the cut of Prolog to implement the corresponding fuzzified predicate. So, for a predicate $student(X)$ will be:

```
f_student(X,V):-
        student(X),!,
        V .=. 1.
f_student(V,0).
```

The result is that $V = 1$ if the goal $pred(X)$ success and $V = 0$ otherwise. With our example we obtain the expected results for $f\_student(peter, V)$ and $f\_student(susan, V)$ but we find the problem in goals as:

```
?- f_student(X,1).
    X = john ? ;
    no
```

where the cut avoids the backtracking and it is impossible to get all the solution. This problem is simply resolve with the alternative transformation:

```
f_student(X,V):-
        if(student(X),V=1,V=0).
```

It resolves the problem of the backtracking because of the implementation of the $if/3$ predicate and returns us

```
?- f_student(X,1).      ?-f_student(susan,V).
    X = john ? ;            .=.(V,0.0)
    X = peter ? ;
    no
```

This is not only useful to give constructive answers to goals of fuzzified predicates but it is the way of getting constructive solutions in fuzzy consults of a fuzzy predicate that is defined combining crisp and fuzzy logic.

6. Examples

6.1. Example 1

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(1) :~ [0.9,1].   small(2) :~ [0.9,1].
small(3) :~ [0.6,0.7]. small(4) :~ [0.3,0.4].
small(5) :~ [0,0.1].   small(6) :~ [0,0.1].
large :# fnot small/2.
```

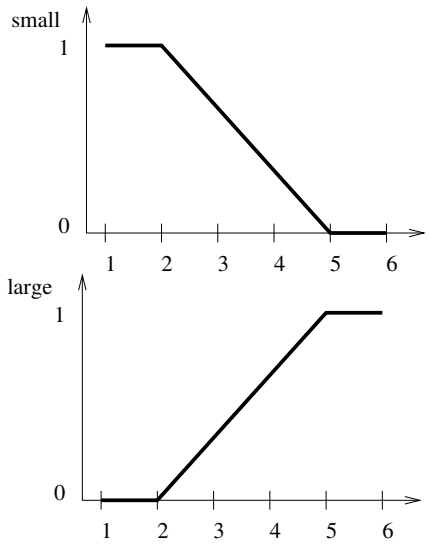


Figure 3. Fuzzy predicates small/2 and large/2

In fuzzy Prolog, this problem can be represented using for example, 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:

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

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

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

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

?- sum(5,V).                 ?- sum(5,V).
   V .=. [0.6,0.7] ?            V .=. [0.66,0.83] ?
   yes                          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.

6.2. Example 2

A more real example could be the problem of compatibility of a couple of turns in a work place. For example teachers that work in different class timetables, telephone operators, etc. Imagine a company where the work is divided in turns of 4 hours per week. Many workers have to combine a couple of turns in the same week and a predicate *compatible*/2 is necessary to check if two turns are compatible or to obtain which couples of turns are compatible. Two turns are compatible when both are correct (working days from Monday to Friday, hours between 8 a.m. and 18 p.m. and there are no repetitions of the same hour in a turn) and in addition when the turns are disjoint.

```
compatible(T1,T2):-
        correct_turn(T1),
        correct_turn(T2),
        disjoint(T1,T2).
```

But there are so many compatible combinations of turns that it would be useful to define the concept of compatibility in a fuzzy way instead of in the crisp way it is defined above. It would express that two turns could be incompatible if one of them is not correct or if they are not disjoint but when they are compatible, they can be more or less compatible. They can have a level of compatibility. Two turns will be more compatible if the working hours are concentrated (the employee has to go to work few days during the week). Also, two turns will be more compatible if there are few free hours between the busy hours of the working days of the timetable.

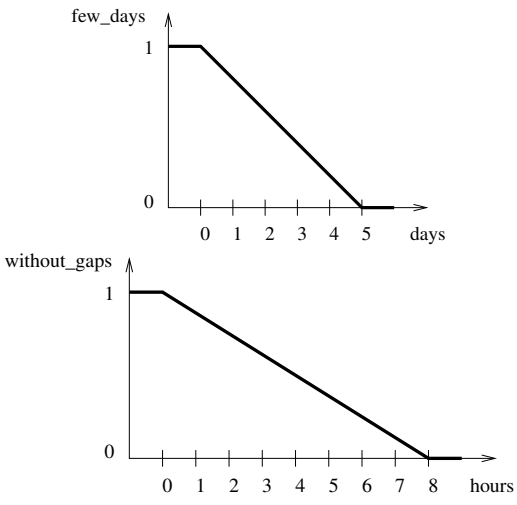


Figure 4. Fuzzy predicates few_days/2 and without_gaps/2

Therefore, we are handing crisp concepts

(*correct_turn*/1, *disjoint*/2) and besides fuzzy concepts (*without_gaps*/2, *few_days*/2). Their definitions, represented in figure 4, are expressed in our language in this simple way:

```
few_days :#
  fuzzy_predicate([(0,1),(1,0.8),
                   (2,0.6),(3,0.4),
                   (4,0.2),(5,0)]).

without_gaps :#
  fuzzy_predicate([(0,1),(1,0.8),
                   (5,0.3),(7,0.1),
                   (8,0)]).
```

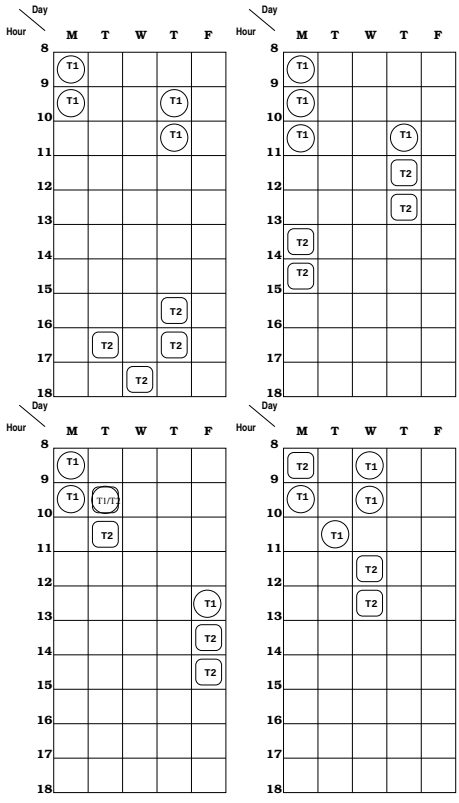


Figure 5. Timetable 1, 2, 3 and 4

A simple implementation combining both types of predicates could be:

```
compatible(T1,T2):~ min
        correct_turn(T1),
        correct_turn(T2),
        disjoint(T1,T2),
        append(T1,T2,T),
        number_of_days(T,D),
        few_days(D),
        number_of_free_hours(T,H),
        without_gaps(H).
```

Where *append*/3 gives the total weekly timetable of 8 hours from joining two turns, *number_of_days*/3 obtains the total number of working days of a weekly timetable and *number_of_free_hours*/2 returns the number of free one hour gaps that the weekly timetable has during the working days.

Observe the timetables in figure 5. We can obtain the compatibility between the couple of turns, T1 and T2, represented in each timetable with the consult $compatible(T1, T2, V)$. The result is $V = 0.2$ for the timetable 1, $V = 0.6$ for the timetable 2. And $V = 0$ for the timetable 3 because the turns are incompatible, and also $V = 0$ for the timetable 4 because turn T2 is not correct.

7. Conclusions and Future work

The novelty of the Fuzzy Prolog presented here 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 [21], 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, but it is also given a way to implement it. Another advantage of our approach is that it can be implemented with little

effort over any other CLP($\mathcal{R}$) system.

We have managed to combine crisp and fuzzy logic in the same compiler. This is a great advantage because it lets us model many problems using fuzzy programs. So we have extended the expressivity of the language and the possibility of applying it to solve real problems.

Presently we are working in several related issues:

- Obtaining constructive answers to negative goals.
- Implementing operators to *defuzzying* truth values.
- Constructing the syntax to work with discrete fuzzy sets.
- Introducing domains of fuzzy sets using types.
- Implementing the expansion over other CLP($\mathcal{R}$) systems.

Acknowledgement

The authors would like to thank the suggestions of Professors Enric Trillas and Francisco Bueno in improving the content and the ideas behind the paper. This work has been partially supported by CICYT (Spain) under projects TIC2000-1420 and TIC99-1151. Finally, the authors would also like to thank the anonymous referees for their valuable comments and suggestions.

REFERENCES

1. C. Vaucheret, S. Guadarrama, S. Muñoz, Fuzzy prolog: A simple general implementation using *clp*(r), in: M. Baaz, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2002, no. 2514 in LNAI, Springer, Tbilisi, Georgia, 2002, pp. 450–463.
2. R. C. T. Lee, Fuzzy logic and the resolution principle, Journal of the Association for Computing Machinery 19 (1) (1972) 119–129.
3. M. Ishizuka, N. Kanai, Prolog-ELF incorporating fuzzy logic, in: IJCAI, 1985, pp. 701–703.
4. J. F. Baldwin, T. P. Martin, B. W. Pilsworth, Fril: Fuzzy and Evidential Reasoning in Artificial Intelligence, John Wiley & Sons, 1995.
5. D. Li, D. Liu, A Fuzzy Prolog Database System, John Wiley & Sons, New York, 1990.
6. Z. Shen, L. Ding, M. Mukaidono, Fuzzy resolution principle, in: Proc. of 18th International Symposium on Multiple-valued Logic, Vol. 5, 1989.
7. F. Klawonn, R. Kruse, A Łukasiewicz logic based Prolog, Mathware & Soft Computing 1 (1) (1994) 5–29.
   URL `citeseer.nj.nec.com/227289.html`
8. E. Y., Shapiro, Logic programs with uncertainties: A tool for implementing rule-based systems, in: IJCAI, 1983, pp. 529–532.
9. S. Bistarelli, U. Montanari, F. Rossi, Semiring-based constraint logic programming: syntax and semantics, in: ACM TOPLAS, Vol. 23, 2001, pp. 1–29.
10. P. Vojtas, Fuzzy logic programming, Fuzzy sets and systems 124 (1) (2001) 361–370.
11. J. Medina, M. Ojeda-Aciego, P. Votjas, Multi-adjoint logic programming with continous semantics, in: LPNMR, Vol. 2173 of Lecture Notes in Computer Science, Springer, Boston, MA (USA), 2001, pp. 351–364.
12. J. Medina, M. Ojeda-Aciego, P. Votjas, A procedural semantics for multi-adjoint logic programming, in: EPIA, Vol. 2258 of Lecture Notes in Computer Science, Springer, Boston, MA (USA), 2001, pp. 290–297.
13. J. Medina, M. Ojeda-Aciego, P. Votjas, A completeness theorem for multi-adjoint logic programming, in: International Fuzzy Systems Conference, IEEE, 2001, pp. 1031–1034.
14. E. Klement, R. Mesiar, E. Pap, Triangular norms, Kluwer Academic Publishers.
15. E. Trillas, S. Cubillo, J. Castro, Conjunction and disjunction on $([0,1],<=)$, Fuzzy Sets and Systems 72 (1995) 155–165.
16. A. Pradera, E. Trillas, T. Calvo, A general class of triangular norm-based aggregation operators: quasi-linear t-s operators, International Journal of Approximate Reasoning 30

(2002) 57–72.
17. H. T. Nguyen, E. A. Walker, A first Course in Fuzzy Logic, Chapman & Hall/Crc, 2000.
18. G. J. Klir, B. Yuan, Fuzzy Sets and Fuzzy Logic, Prentice Hall, 1995.
19. S. C. E. Trillas, A. Pradera, A mathematical model for fuzzy connectives and its application to operators behavioural study, Vol. 516, Kluwer Academic Publishers (Series: The Kluwer International Series in Engineering and Computer Sciences), 1999, Ch. 4, pp. 307–318.
20. D. Dubois, H. Prade, A review of fuzzy set aggregation connectives, Information Sciences 36 (1985) 85–121.
21. L. Zadeh, Fuzzy sets as a basis for a theory of possibility, Fuzzy sets and systems 1 (1) (1978) 3–28.
22. A. Tarski, A lattice-theoretical fixpoint theorem and its applications, Pacific Journal of Mathematics 5 (1955) 285–309.
23. J. Jaffar, J. L. Lassez, Constraint Logic Programming, in: ACM Symp. Principles of Programming Languages, ACM, 1987, pp. 111–119.
24. J. Jaffar, S. Michaylov, P. J. Stuckey, R. H. C. Yap, The clp(r) language and system, ACM Transactions on Programming Languages and Systems 14 (3) (1992) 339–395.
25. D. Cabeza, M. Hermenegildo, A New Module System for Prolog, in: CL2000, no. 1861 in LNAI, Springer-Verlag, 2000, pp. 131–148.
26. K. L. Clark, Negation as failure, in: J. M. H. Gallaire (Ed.), Logic and Data Bases, New York, NY, 1978, pp. 293–322.
27. S. Muñoz, C. Vaucheret, S. Guadarrama, Combining crisp and fuzzy logic in a prolog compiler, in: J. Moreno, J. Mariño (Eds.), Joint Conference on Declarative Programming: APPIA-GULP-PRODE 2002, Madrid, Spain, 2002, pp. 23–38.