

Incomplete Fuzzy Information in Prolog

S. Munoz-Hernandez

Universidad Politécnica de Madrid
Spain
susana@fi.upm.es

C. Vaucheret

Universidad Nacional del Comahue
Argentina
vaucheret@ibap.com.ar

Abstract

Incomplete information is a problem in many aspects of actual environments. In many sceneries the knowledge is not represented in a crisp way. It is common to find fuzzy concepts or problems with some level of uncertainty. It is difficult to find practical systems which handle fuzziness and uncertainty and the few examples that we can find are minority. To extend a popular system (which many of programmers are using) with this ability seems to be an interesting issue.

Our first work (Fuzzy Prolog [1]) was a language that models $\mathcal{B}([0, 1])$ -valued Fuzzy Logic. In the Borel Algebra, $\mathcal{B}([0, 1])$, truth value is represented using unions of intervals of real numbers. It subsumed former approaches because it was more general in truth value representation and propagation than them.

Now, we enhance our former approach by using default knowledge to represent incomplete information in Logic Programming. We also provide the implementation of this new framework. This new release of Fuzzy Prolog handles incomplete information and it has a complete semantics (the before one was incomplete as Prolog) which we discuss. New Fuzzy Prolog is more expressive to represent real world.

Keywords: Incomplete knowledge, Modeling Uncertainty, Fuzzy Logic Programming, Constraint Programming Application, Fuzzy Prolog Implementation.

This research was partly supported by the Spanish MCYT project TIC2003-01036 and the project AL05_PID_0040.

1 Introduction

World information is not represented in a crisp way. Its representation is imperfect, fuzzy, etc., so that, the management of uncertainty is very important in knowledge representation. There are multiple frameworks for incorporating uncertainty in logic programming (Fuzzy set theory, Probability theory, Multi-valued logic, Possibilistic logic, etc.)

In [4] it was proposed a general framework that generalizes many of the precedent approaches. At the same time in [6] was provided an analogous theoretical framework and a prototype for Prolog was implemented. Basically, a rule is of the form A, B_1, \dots, B_n , where the assignment I of certainties is taken from a certainty lattice, to the B_i s. The certainty of A is computed by taking the set of the certainties $I(B_i)$ and then they are propagated using the function F that is an aggregation operator. This is a very flexible approach and in [7, 1] practical examples in a Prolog framework are presented.

In this work we extend the approach of [7] with arbitrary assignments of default certainty values (non-uniform default assumptions). The usual semantics of logic programs can be obtained through a unique computation method, but using different assumptions in a uniform way to assign the same default truth-value to all the atoms.

The rest of the paper is organized as follows. Section 2 introduces the Fuzzy Prolog language. A complete description of the new semantics is provided in Section 3. Finally, we conclude and discuss some future work (Section 4).

2 Fuzzy Prolog

Fuzzy Prolog is more general than previous approaches in some respects:

1. A truth value will be a finite union of sub-intervals on $[0, 1]$. This is represented by Borel Algebra, $\mathcal{B}([0, 1])$, while the Algebra $\mathcal{E}([0, 1])$ only considers intervals.
2. A truth value will be propagated through the rules by means of an *aggregation operator*. 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 $f(0, \dots, 0) = 0$ and $f(1, \dots, 1) = 1$, and in addition it is monotonic and continuous, then it is called *aggregation operator*. If we deal with the definition of fuzzy sets as intervals it is necessary to generalize from *aggregation operators* of numbers to *aggregation operators* of intervals.
3. The declarative and procedural semantics for Fuzzy Logic programs are given and their equivalence is proven.
4. An implementation of the proposed language is presented. A *fuzzy program* is a finite set of: *fuzzy facts*, $A \leftarrow v$, where A is an atom and v , a truth value, is an element in $\mathcal{B}([0, 1])$ and *fuzzy clauses*, $A \leftarrow_F B_1, \dots, B_n$, where A, B_1, \dots, B_n are atoms, and F is an interval-aggregation operator, which induces a union-aggregation, \mathcal{F} of truth values in $\mathcal{B}([0, 1])$. We obtain information from the program through *fuzzy queries* or *fuzzy goals*, $v \leftarrow A ?$, where A is an atom, and v is a variable, possibly instantiated, that represents a truth value in $\mathcal{B}([0, 1])$.

In [1] (and furthermore 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 c is *consistent*, and is denoted as *solvable*(c). Consult [1] for more details.

3 Semantics

This section contains a reformulation of the semantics of Fuzzy Prolog. It is now complete thanks to the inclusion of default value.

3.1 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 3.1 (default value) *We assume there is a function default which implement the Default Knowledge Assumptions. It assigns an element of $\mathcal{B}([0, 1])$ to each element of the Herbrand Base. If the Closed World Assumption is used, then $\text{default}(A) = [0, 0]$ for all A in Herbrand Base. If Open World Assumption is used instead, $\text{default}(A) = [0, 1]$ for all A in Herbrand Base.*

Definition 3.2 (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) a truth value, in $\mathcal{B}([0, 1])$, to each element of B_I , or
 - (b) $\text{default}(A)$, if A does not belong to B_I .

Definition 3.3 (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 3.4 (Borel inclusion \subseteq_{BI}) *Given two unions of intervals $U = I_1 \cup \dots \cup I_N$, $U' = I'_1 \cup \dots \cup I'_M$ in $\mathcal{B}([0, 1])$, $U \subseteq_{BI} U'$ if and only if $\forall I_i \in U$, $i \in 1..N$, $\exists I_{i1}, \dots, I_{iL}$ intervals such that $I_{i1} \cup \dots \cup I_{iL} = I_i$, $I_{i1} \cap \dots \cap I_{iL} = \emptyset$ and for all $k \in 1..L$, $\exists I'_{jk} \in U'$. $I_{ik} \subseteq_{II} I'_{jk}$ where $jk \in 1..M$.*

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.

Notice that we have redefined interpretation and Borel inclusion with respect to the definitions in [1]. We will also redefine operational semantics and therefore internal implementation of the Fuzzy Prolog library. Sections 3 and 4 are completely new too. For uniformity reasons we have kept the same syntax that was used in [1] in fuzzy programs.

Definition 3.5 (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 3.6 (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 3.7 (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 $A \leftarrow_F B_1, \dots, B_n$ when the following holds: for all valuation σ , $\sigma(A) \in B_I$ and $v \subseteq_{BI} V_I(\sigma(A))$, where $v = \mathcal{F}(V_I(\sigma(B_1)), \dots, V_I(\sigma(B_n)))$ and \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.

Theorem 3.1 (model intersection property)
 Let $I_1 = \langle B_{I_1}, V_{I_1} \rangle$, $I_2 = \langle B_{I_2}, V_{I_2} \rangle$ 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 $A \leftarrow_F B_1, \dots, B_n$ in P
 - since $\sigma(A) \in B_{I_1}$ and $\sigma(A) \in B_{I_2}$, then $\sigma(A) \in B_{I_1} \cap B_{I_2} = B_M$.
 - if $v = \mathcal{F}(V_M(\sigma(B_1)), \dots, V_M(\sigma(B_n)))$, 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 $A \leftarrow_F B_1, \dots, B_n$

and M is model of P .

Remark 3.1 (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 $lm(P)$.

3.2 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 $lm(P)$. We include it here for clarity reasons although it is the same that in [1].

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

where

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

Note that since I' must be an interpretation, $V_{I'}(A) = default(A)$ for all $A \notin B_{I'}$.

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 :

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

and dually,

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

Since the first limit ordinal is 0, it follows that in particular, $G \uparrow 0 = \perp_X$ (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 $lfp(T_P) = T_P \uparrow \omega$.

Lemma 3.1 *Let P a fuzzy program, M is a model of P if and only if M is a pre-fixpoint 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 Herbrand Base, if $A \in B_{T_P}$, 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 , $A \leftarrow_F A_1, \dots, A_n$ where $\{A_1, \dots, A_n\} \subseteq$

B_M and $v = \mathcal{F}(V_M(A_1), \dots, V_M(A_n))$. 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 $T_P(M) \sqsubseteq M$. \square . If $A \notin B_{T_P}$ then $V_{T_P}(A) = default(A) \subseteq_{BI} V_M(A)$.

Analogously, for the “only if” direction, for each ground instance

$v = \mathcal{F}(V_M(A_1), \dots, V_M(A_n))$, $A \in B_{T_P}$ and $v \subseteq_{BI} V_{T_P}(A)$, but as $T_P(M) \sqsubseteq M$, $B_{T_P} \subseteq B_M$ 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 . \square

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 .

Theorem 3.2 *Let P be a fuzzy program, $lm(P) = lfp(T_P)$.*

Proof.

$$\begin{aligned} lm(P) &= \bigcap \{M \mid M \text{ is a model of } P\} \\ &= \bigcap \{M \mid M \text{ is a pre-fixpoint of } P\} \\ &\quad \text{from lemma 3.1} \\ &= lfp(T_P) \quad \text{by the Knaster-Tarski} \\ &\quad \text{Fixpoint Theorem [5]}\square \end{aligned}$$

3.3 Operational Semantics

The improvement of Fuzzy Prolog is significative in its new procedural semantics that 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 $\text{solvable}(S \wedge \mu_a = v)$.
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 $\text{solvable}(S \wedge c)$.
3. $\langle A \cup a, \sigma, S \rangle \rightarrow \langle A, \sigma, S \wedge \mu_a = v \rangle$, if none of the above are applicable and $\text{solvable}(S \wedge \mu_a = v)$ where $\mu_a = \text{default}(a)$.

The success set $SS(P)$ collects the answers to simple goals $p(\hat{x})$. We define $SS(P) = \langle B, V \rangle$ where $B = \{p(\hat{x})\sigma \mid \langle p(\hat{x}), \emptyset, \text{true} \rangle \rightarrow^* \langle \emptyset, \sigma, S \rangle\}$ is the set of elements of the Herbrand Base that are instantiated and that have succeeded; and $V(p(\hat{x})) = \cup\{v \mid \langle p(\hat{x}), \emptyset, \text{true} \rangle \rightarrow^* \langle \emptyset, \sigma, S \rangle, \text{ and } v \text{ is the solution of } S\}$ 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 $p(\hat{x})$ is computed.

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 3.8 (Breadth-first transition)

Given the following set of valid transitions:

$$\begin{aligned} &\langle \{A_1, \dots, A_n\}, \sigma, S \rangle \rightarrow \\ &\quad \langle \{A_2, \dots, A_n\} \cup B_1, \sigma \cdot \theta_1, S \wedge c_1 \rangle \\ &\langle \{A_1, \dots, A_n\}, \sigma, S \rangle \rightarrow \\ &\quad \langle \{A_1, A_3, \dots, A_n\} \cup B_2, \sigma \cdot \theta_2, S \wedge c_2 \rangle \\ &\langle \{A_1, \dots, A_n\}, \sigma, S \rangle \rightarrow \\ &\quad \langle \{A_1, \dots, A_{n-1}\} \cup B_n, \sigma \cdot \theta_n, S \wedge c_n \rangle \end{aligned}$$

a breadth-first transition is defined as

$$\langle \{A_1, \dots, A_n\}, \sigma, S \rangle \rightarrow_{BF} \langle B_1 \cup \dots \cup B_n, \sigma \cdot \theta_1 \cdot \dots \cdot \theta_n, S \wedge c_1 \wedge \dots \wedge c_n \rangle$$

in which all literals are reduced at one step.

Theorem 3.3 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 length less or equal to $n+1$ for a program P , $\langle \{A_1, \dots, A_k\}, \sigma, S_1 \rangle \rightarrow_{BF}^* \langle \emptyset, \theta, S_2 \rangle$ iff $A_i\theta \in B_{T_{P_n}}$ and $\text{solvable}(S \wedge \mu_{A_i} = v_i)$ and $v_i \subseteq_{BI} V_{T_{P_n}}(A_i\theta)$ for all $i \in \{1, \dots, k\}$.

Proof. The proof is by induction on n . For the base case, all the literals are reduced using the first type of transitions or the last one, that is, for each literal A_i , it exists 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 $\text{solvable}(S_1 \wedge \mu_{A_i} = v_i)$ or $\mu_{A_i} = \text{default}(A_i)$. By definition of T_P , 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,

$$\begin{aligned} &\langle \{A_1, \dots, A_k\}, \sigma_1, S_1 \rangle \rightarrow_{BF} \langle B, \sigma_2, S_2 \rangle \rightarrow_{BF} \\ &\dots \rightarrow_{BF} \langle \emptyset, \sigma_n, S_n \rangle \\ &\text{the transition } \langle \{A_1, \dots, A_k\}, \sigma_1, S_1 \rangle \rightarrow_{BF} \\ &\langle B, \sigma_2, S_2 \rangle \end{aligned}$$

When a literal A_i is reduced using a fact or there is not rule for A_i the result is the same as in the base case, otherwise there is a clause $h_i \leftarrow_F B_{1_i}, \dots, B_{m_i}$ 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 $\text{solvable}(S_2 \wedge \mu_{B_{j_i}} = v_{j_i})$ 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 $\text{solvable}(S_1 \wedge \mu_{A_i} = v_i)$ and $v_i \subseteq_{BI} V_{T_{P_n}}(A_i\sigma_1)$. \square

Theorem 3.4 For a program P there is a successful derivation $\langle p(\hat{x}), \emptyset, \text{true} \rangle \rightarrow^* \langle \emptyset, \sigma, S \rangle$ iff $p(\hat{x})\sigma \in B$ and v is the solution of S and $v \subseteq_{BI} V(p(\hat{x})\sigma)$ where $\text{lfp}(T_P) = \langle B, V \rangle$

Proof. It follows from the fact that $\text{lfp}(T_P) = T_P \uparrow \omega$ and from the Theorem 3.3. \square

Theorem 3.5 For a fuzzy program P the three semantics are equivalent, i.e.

$$SS(P) = \text{lfp}(TP) = \text{lm}(P)$$

Proof. the first equivalence follows from Theorem 3.4 and the second from Theorem 3.2. \square

4 Conclusions and Future work

Constraint Logic Programming 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}) [3] has linear arithmetic constraints and computes over the real numbers.

Fuzzy Prolog was implemented in [1] as a syntactic extension of a CLP(\mathcal{R}) system. CLP(\mathcal{R}) was incorporated as a library in the Ciao Prolog system¹. The *fuzzy* library (or *package* in the Ciao Prolog terminology) which implements the interpreter of our fuzzy Prolog language has been modified to handle default reasoning.

Fuzzy Prolog presented in [1] 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 [8], and after that, translating fuzzy predicates into CLP(\mathcal{R}) clauses. The rest of the computation is resolved by the compiler.

In this paper we propose to enrich a practical and extended language (Prolog) with more expressivity by adding default reasoning and fore-
 head the possibility of handling incomplete information. We have developed a complete and sound semantics for handling incomplete fuzzy information and we have also provided a real implementation based in our former Fuzzy Prolog approach. So we have extended the expressivity of the language and the possibility of applying it to solve real problems in which the information can be defined, fuzzy or incomplete.

Presently we are working in several related issues: obtaining constructive answers to negative goals, constructing the syntax to work with discrete fuzzy sets and its applications, implementing a representation model using unions instead of

using backtracking, introducing domains of fuzzy sets using types, implementing the expansion over other CLP(\mathcal{R}) systems.

References

- [1] S. Guadarrama, S. Muñoz, and C. Vaucheret. Fuzzy prolog: A new approach using soft constraints propagation. *Fuzzy Sets and Systems, FSS*, 144(1):127–150, 2004.
- [2] M. Hermenegildo, F. Bueno, D. Cabeza, M. Carro, M. García de la Banda, P. López-García, and G. Puebla. The CIAO Multi-Dialect Compiler and System: An Experimentation Workbench for Future (C)LP Systems. In *Parallelism and Implementation of Logic and Constraint Logic Programming*, pages 65–85. Nova Science, Commack, NY, USA, April 1999.
- [3] 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.
- [4] J. Medina, M. Ojeda-Aciego, and P. Votjas. Similarity-based unification: a multi-adjoint approach. *Fuzzy Set and Systems*, 146(1):43–62, August 2004. Selected Papers from EUSFLAT 2001.
- [5] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
- [6] C. Vaucheret, S. Guadarrama, and S. Muñoz. Fuzzy prolog: A simple implementation using clp(r). In *Constraints and Uncertainty*, Paphos (Cyprus), 2001. CP'2001 workshop.
- [7] C. Vaucheret, S. Guadarrama, and S. Muñoz. Fuzzy prolog: A simple general implementation using clp(r). In M. Baaz and A. Voronkov, editors, *LPAR Proceedings*, number 2514 in LNAI, pages 450–463, Tbilisi, Georgia, October 2002. Springer-Verlag.
- [8] L. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy sets and systems*, 1(1):3–28, 1978.

¹The Ciao system [2] including our Fuzzy Prolog implementation can be downloaded from <http://www.clip.dia.fi.upm.es/Software/Ciao>.