Multi-adjoint Fuzzy Logic Programming with Defaults and Constructive Answers

Hannes Strass, Susana Munoz-Hernandez, and Víctor Pablos Ceruelo

Universidad Politécnica de Madrid **
hannes.strass@alumnos.upm.es, {susana,vpablos}@fi.upm.es
http://babel.ls.fi.upm.es/

Abstract. In this paper we present the semantics of RFuzzy, a fuzzy Logic Programming framework that represents truth values using real numbers from the unit interval. RFuzzy provides some useful extensions: default values to represent missing information, and typed arguments to intuitively restrict predicate domains. Together, they allow the system to provide constructive answers when querying for individuals satisfying constraints over the truth value in addition to the classical queries over the own truth value.

Key words: Fuzzy reasoning, Semantics, Implementation tool, Fuzzy Logic, Multi-adjoint logic, Logic Programming Application

1 Introduction

Logic programming has been successfully used in knowledge representation and reasoning for decades. However, a serious drawback is the lack of an easy and user-friendly way to represent vague world information.

To address this issue, multiple frameworks for incorporating uncertainty in logic have been developed over the years: fuzzy set theory, probability theory, multi-valued logic, or possibilistic logic; to mention only some.

The result of introducing fuzzy logic into logic programming has been the development of several fuzzy systems over Prolog. These systems replace the inference mechanism, SLD-resolution, of Prolog with a fuzzy variant that 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, the Fril Prolog system and the F-Prolog language. However, there is no common method for fuzzifying Prolog as has been noted in [8].

Over the last few years several papers have been published by Medina et al. [4, 3] about multi-adjoint logic programming. The theoretical model described in these works led to the development of FLOPER [5], another fuzzy logic programming system. It has a logic-programming-inspired syntax and provides free

** This work is partially supported by the project DESAFIOS - TIN 2006-15660-C02-02 from the Spanish Ministry of Education and Science, by the Spanish Ministry of Science and Innovation Research Staff Training Program - BES-2008-008320 and by the project PROMESAS - S-0505/TIC/0407 from the Madrid Regional Government.

choice of aggregation operators and credibility of rules just as RFuzzy does. There are however some things that FLOPER cannot do:

- deal with missing information (using default truth value declarations),
- type atoms and predicates to give constructive answers, and
- provide syntactic sugar to express truth value functions.

One of the most promising fuzzy tools for Prolog was the "Fuzzy Prolog" system [10, 1]. This approach is more general than others in some respects:

1. A truth value is a finite union of sub-intervals on $[0, 1]$.
2. A truth value is propagated through the rules by means of an *aggregation operator*. The definition of *aggregation operator* is general.
3. Crisp and fuzzy reasoning are consistently combined [6].

Fuzzy Prolog adds fuzziness to a Prolog compiler using CLP($\mathcal{R}$) instead of implementing a new fuzzy resolution as other former fuzzy Prologs do. So, it uses Prolog's built-in inference mechanism, and the constraints and their operations provided by CLP($\mathcal{R}$) to handle the concept of partial truth. It represents intervals as constraints over real numbers and *aggregation operators* as operations with these constraints.

There are other proposals, e.g. in [7], that provide an interpretation of truth values as intervals, but Fuzzy Prolog proposed to generalize this concept to unions of intervals for the first time.

The generality of the approach pursued in [10, 1] turned out to make many users feel uncomfortable: Fuzzy Prolog is rather expressive, so it is not always clear how knowledge should be represented. Furthermore, interpreting the output that comes as a sequence of constraints maybe possible for a human but is very hard to do for a computer program – especially basing a decision upon it may not be straightforward.

To address these issues, we propose the RFuzzy framework. It is considerably simpler to use than the above-mentioned Fuzzy Prolog, but still contains many of its nice features. In RFuzzy, truth values will be represented by real numbers from the unit interval $[0, 1]$. This simplifies making modelling decisions and interpreting the output of the system. We still use the general concept of aggregation operators to be able to model different fuzzy logics.

The rest of the paper is organized as follows: Section 2 introduces the abstract syntax of RFuzzy. In Section 3 we present two different semantics for RFuzzy and sketch the proof of their equivalence. Finally, we conclude in the last section.

2 RFuzzy Syntax

We use a signature Σ of function symbols and a set of variables V to "build" the *term universe* $\mathrm{TU}_{\Sigma,V}$ (whose elements are the *terms*). It is the minimal set such that each variable is a term and terms are closed under Σ-operations. In particular, constant symbols are terms.

Similarly, with use a signature Π of predicate symbols to define the *term base* $\mathrm{TB}_{\Pi,\Sigma,V}$ (whose elements are called *atoms*). Atoms are predicates whose arguments are elements of $\mathrm{TU}_{\Sigma,V}$. Atoms and terms are called *ground* if they do not contain variables. As usual, the *Herbrand universe* H is the set of all ground terms, and the *Herbrand base* B is the set of all atoms with arguments from the Herbrand universe.

To combine truth values in the set of real truth values $[0,1]$, we make use of *aggregation operators.* A function $\hat{F} : [0,1]^n \to [0,1]$ is called an aggregation operator if it verifies $\hat{F}(0,\ldots,0) = 0$ and $\hat{F}(1,\ldots,1) = 1$. We use the signature Ω to denote the set of used operator symbols F and $\hat{\Omega}$ to denote the set of their associated aggregation operators $\hat{F}$. An n-ary aggregation operator is called *monotonic in the i-th argument*, if additionally $x \le x'$ implies $\hat{F}(x_1,\ldots,x_{i-1},x,x_{i+1},\ldots,x_n) \le \hat{F}(x_1,\ldots,x_{i-1},x',x_{i+1},\ldots,x_n)$. An aggregation operator is called *monotonic* if it is monotonic in all arguments.

Immediate examples for aggregation operators that come to mind are typical examples of t-norms and t-conorms: minimum $\min(a,b)$, maximum $\max(a,b)$, product $a \cdot b$, and probabilistic sum $a + b - a \cdot b$.

The above general definition of aggregation operators subsumes however all kinds of minimum, maximum or mean operators.

Definition 1. *Let Ω be an aggregation operator signature, Π a predicate signature, Σ a term signature, and V a set of variables.*

A fuzzy clause *is written as*

$$A \xleftarrow{c,F_c}_F B_1,\ldots,B_n$$

where $A \in \mathrm{TB}_{\Pi,\Sigma,V}$ is called the head, $B_1,\ldots,B_n \in \mathrm{TB}_{\Pi,\Sigma,V}$ is called the body, $c \in [0,1]$ is the credibility value, and $F_c \in \Omega^{(2)}$ and $F \in \Omega^{(n)}$ are aggregation operator symbols (for the credibility value and the body resp.)

A fuzzy fact *is a special case of a clause where $n = 0$, $c = 1$, F_c is the usual multiplication of real numbers "$\cdot$" and $F = v \in [0,1]$. It is written as $A \leftarrow v$.*

A fuzzy query *is a pair $\langle A, v\rangle$, where $A \in \mathrm{TB}_{\Pi,\Sigma,V}$ and v is either a "new" variable that represents the initially unknown truth value of A or it is a concrete value $v \in [0,1]$ that is asked to be the truth value of A.*

Intuitively, a clause can be read as a special case of an implication: we combine the truth values of the body atoms with the aggregation operator associated to the clause to yield the truth value for the head atom. For this truth value calculation we are completely free in the choice of an operator.

Example. Consider the following clause, that models to what extent cities can be deemed good travel destinations – the quality of the destination depends on the weather and the availability of sights:

$$\mathsf{good\text{-}destination(X)} \xleftarrow{1.0,\cdot}_{\cdot} \mathsf{nice\text{-}weather(X),\ many\text{-}sights(X).}$$

The credibility value of the rule is 1.0, which means that we have no doubt about this relationship. The aggregation operator used here in both cases is the product "$\cdot$". We enrich the knowledge base with facts about some cities and their continents:

$$\begin{gathered}\textsf{nice-weather(madrid)} \leftarrow 0.8, \textsf{nice-weather(moscow)} \leftarrow 0.2,\\ \textsf{many-sights(madrid)} \leftarrow 0.6, \textsf{many-sights(sydney)} \leftarrow 0.6,\\ \textsf{city-continent(madrid, europe)} \leftarrow 1.0, \textsf{city-continent(sydney, australia)} \leftarrow 1.0,\end{gathered}$$

It can be seen that no information about the weather in Sydney or sights in Moscow is available although these cities are "mentioned".

In the above example, the knowledge that we represented using fuzzy clauses and facts was not only vague but moreover incomplete. As this is rather the norm than the exception, we would like to have a mechanism that can handle non-present information.

In standard logic programming, the closed-world assumption is employed, i.e. the knowledge base is not only assumed to be sound but moreover to be complete. Everything that can not be derived from the knowledge is assumed to be false. This could be easily modelled in this framework by assuming the truth value 0 as "default" truth value, so to speak. Yet we want to pursue a slightly more general approach: arbitrary default truth values will be explicitly stated for each predicate. We even allow the definition of different default truth values for different arguments of a predicate. This is formalized as follows.

Definition 2. *A* default value declaration *for a predicate $p \in \Pi^{(n)}$ is written as* `default`$(p(X_1, \dots, X_n)) = [\delta_1 \textit{ if } \varphi_1, \dots, \delta_m \textit{ if } \varphi_m]$ *where $\delta_i \in [0,1]$ for all i. The φ_i are first-order formulas restricted to terms from $\mathrm{TU}_{\Sigma,\{X_1,\dots,X_n\}}$, the predicates $=$ and $\neq$, the symbol* true*, and the junctors $\wedge$ and $\vee$ in their usual meaning.*

Example (continued). Let us add the following default value declarations to the knowledge base and thus close the mentioned gaps.

$$\begin{gathered}\texttt{default}(\textsf{nice-weather(X)}) = 0.5,\\ \texttt{default}(\textsf{many-sights(X)}) = 0.2,\\ \texttt{default}(\textsf{good-destination(X)}) = 0.3\end{gathered}$$

They could be interpreted as: when visiting an arbitrary city of which nothing further is known, it is likely that you have nice weather but you will less likely find many sights. Irrespective of this, it will only to a small extent be a good travel destination.

To model the fact that a city is not on a continent unless stated otherwise, we add another default value declaration for city-continent:

$$\texttt{default}(\textsf{city-continent(X, Y)}) = 0.0.$$

Notice that in this example $m = 1$ and $\varphi_1 =$ true for all the default value declarations.

The default values allow our knowledge base to answer arbitrary questions about predicates that occur in it. But will the answers always make sense? To stay in the above example, if we ask a question like "What is the truth value of nice-weather(australia)?" we will get the answer "0.5" which does not make too much sense since Australia is not a city, but a continent.

To resolve this, we introduce types into the language. Types can be viewed as inherent properties of terms – each term can have zero or more types. We use them to restrict the domains of predicates.

Definition 3. *A* term type declaration *assigns a type $\tau \in \mathcal{T}$ to a term $t \in H$ and is written as $t : \tau$. A* predicate type declaration *assigns a type $(\tau_1, \dots, \tau_n) \in \mathcal{T}^n$ to a predicate $p \in \Pi^n$ and is written as $p : (\tau_1, \dots, \tau_n)$, where τ_i is the type of p's i-th argument. For a set of type declarations, the* type mapping $\mathfrak{t}_T : H \to 2^{\mathcal{T}}$ *is defined as $\mathfrak{t}_T(t) := \{\tau \in \mathcal{T} \mid (t : \tau) \in T\}$.*

Example (continued). Using the set of types $\mathcal{T} = \{\mathsf{City}, \mathsf{Continent}\}$, we add some term type declarations to our knowledge base:

$$\mathsf{madrid} : \mathsf{City}, \mathsf{istanbul} : \mathsf{City}, \mathsf{sydney} : \mathsf{City}, \mathsf{moscow} : \mathsf{City};$$
$$\mathsf{antarctica} : \mathsf{Continent}, \mathsf{europe} : \mathsf{Continent}.$$

We also type the predicates in the obvious way:

$$\mathsf{nice\text{-}weather} : (\mathsf{City}), \mathsf{many\text{-}sights} : (\mathsf{City}), \mathsf{good\text{-}destination} : (\mathsf{City}),$$
$$\mathsf{city\text{-}continent} : (\mathsf{City}, \mathsf{Continent}).$$

For a ground atom $A = p(t_1, \dots, t_n) \in B$ we say that it is *well-typed with respect to T* iff $p : (\tau_1, \dots, \tau_n) \in T$ implies $\tau_i \in \mathfrak{t}_T(t_i)$ for all i.

For a ground clause $A \xleftarrow{c, F_c}_F B_1, \dots, B_n$ we say that it is well-typed w.r.t. T iff all B_i are well-typed for $1 \le i \le n$ implies that A is well-typed (i.e. if the clause preserves well-typing). We say that a non-ground clause is well-typed iff all its ground instances are well-typed.

Example (continued). With respect to the given type declarations, city-continent(moscow, antarctica) is well-typed whereas city-continent(asia, europe) is not.

A *fuzzy logic program P* is a triple $P = (R, D, T)$ where R is a set of fuzzy clauses, D is a set of default value declarations, and T is a set of type declarations.

Definition 4. *A fuzzy logic program $P = (R, D, T)$ is called* well-defined *iff*

- *for each predicate symbol p/n occurring in R, there exist both a predicate type declaration and a default value declaration;*
- *all clauses in R are well-typed;*
- *for each default value declaration* $\texttt{default}(p(X_1, \dots, X_n)) = [\delta_1 \text{ if } \varphi_1, \dots, \delta_m \text{ if } \varphi_m]$, *the formulas φ_i are pairwise contradictory and $\varphi_1 \vee \dots \vee \varphi_m$ is a tautology, i.e. exactly one default truth value applies to each element of p/n's domain.*

3 RFuzzy Semantics

The possibility to define default truth values for predicates offers us a great deal of flexibility and expressivity. But it also has its drawbacks: reasoning with defaults is inherently non-monotonic – we might have to withdraw some conclusions that have been made in an earlier stage of execution. To capture this formally, we attach to each truth value a symbol that indicates how this value has been concluded. There are 3 different cases: a truth value can be determined

- exclusively by application of program facts and clauses; $\blacktriangledown$
- by indirect use of default values; $\blacklozenge$
- directly via a default value declaration. $\blacktriangle$

Each of these cases will be indicated with the respective symbol.

We need to be able to compare the attributes (in order to be able to prefer one conclusion over another) and to combine them to keep track of default value usage in the course of computation. This is formalized by setting the ordering $<_a$ on truth value attributes such that $\blacktriangle <_a \blacklozenge <_a \blacktriangledown$.

The operator $\circ : \{\blacktriangle, \blacklozenge, \blacktriangledown\} \times \{\blacktriangle, \blacklozenge, \blacktriangledown\} \to \{\blacktriangle, \blacklozenge, \blacktriangledown\}$ is then defined as:

$$x \circ y := \begin{cases} \blacktriangledown & \text{if } x = y = \blacktriangledown \\ \blacktriangle & \text{if } x = y = \blacktriangle \\ \blacklozenge & \text{otherwise} \end{cases}$$

It is designed to keep track of attributes during computation: only when two "safe" truth values are combined, the result is known to be "safe", in all other cases it is "unsafe". It should be noted that "$\circ$" is monotonic.

The truth values that we use in the description of the semantics will be real values $v \in [0, 1]$ with an attribute (i.e. a $z \in \{\blacktriangle, \blacklozenge, \blacktriangledown\}$) attached to it. We will write them as zv. The ordering $\preccurlyeq$ on the truth values will be the lexicographic product of $<_a$, the ordering on the attributes, and the standard ordering $<$ of the real numbers. The set of truth values is thus totally ordered as follows:

$$\perp \prec \blacktriangle 0 \prec \cdots \prec \blacktriangle 1 \prec \blacklozenge 0 \prec \cdots \prec \blacklozenge 1 \prec \blacktriangledown 0 \prec \cdots \prec \blacktriangledown 1.$$

3.1 Least model semantics

A *valuation* $\sigma : V \to B$ is an assignment of ground terms to variables. Each valuation σ uniquely constitutes a mapping $\hat{\sigma} : \mathrm{TU}_{\Sigma,V} \to B$ that is defined in the obvious way.

A *fuzzy Herbrand interpretation* (or short, *interpretation*) of a fuzzy logic program is a mapping $I : B \to \mathbb{T}$ that assigns truth values to ground atoms.

The *domain* of an interpretation is the set of all atoms to which a "proper" truth value is assigned: $\mathsf{Dom}(I) := \{A \mid A \in B, I(A) \succ \perp\}$.

For two interpretations I and J, we say I *is less than or equal to* J, written $I \sqsubseteq J$, if $I \sqsubseteq J$ iff $I(A) \preccurlyeq J(A)$ for all $A \in B$.

Accordingly, the infimum (or intersection) and supremum (or union) of interpretations are, for all $A \in B$, defined as $(I \sqcap J)(A) := \min(I(A), J(A))$ and $(I \sqcup J)(A) := \max(I(A), J(A))$.

The pair $(\mathcal{I}_P, \sqsubseteq)$ of the set of all interpretations of a given program with the interpretation ordering forms a complete lattice. This follows readily from the fact that the underlying truth value set $\mathbb{T}$ forms a complete lattice with the truth value ordering $\preccurlyeq$.

Definition 5 (Model). *Let $P = (R, D, T)$ be a fuzzy logic program.*

We say that I is a model of the clause $r \in R$ *and write* $I \Vdash A \xleftarrow{c,F_c}_F B_1, \ldots, B_n$ *iff for all valuations σ, we have: if $I(\sigma(B_i)) = z_i v_i \succ \bot$ for all i, then $I(\sigma(A)) \succcurlyeq z'v'$ where $z' = z_1 \circ \cdots \circ z_n$ and $v' = \hat{F}_c(c, \hat{F}(v_1, \ldots, v_n)))$.*

We say that I is a model of the default value declaration $d \in D$ *and write*

$$I \Vdash \texttt{default}(p(X_1, \ldots, X_n)) = [\delta_1 \text{ if } \varphi_1, \ldots, \delta_m \text{ if } \varphi_m]$$

iff for all valuations σ, we have: if $\sigma(p(X_1, \ldots, X_n))$ is well-typed w.r.t. T, then there exists $1 \le j \le m$ such that $\sigma(\varphi_j)$ holds and $I(\sigma(p(X_1, \ldots, X_n))) \succcurlyeq \blacktriangle\delta_j$.

Consequently, $I \Vdash R$ iff $I \Vdash r$ for all $r \in R$, and $I \Vdash D$ iff $I \Vdash d$ for all $d \in D$.

Finally, we say that I is a model of the program P, *$I \Vdash P$, iff $I \Vdash R$ and $I \Vdash D$.*

The following proposition will be an important prerequisite to define the least model semantics. It states that the infimum (or intersection) of two models of a program will again be a model. The existence of a least model is then obvious and easily defined as the intersection of all models.

Proposition 6 (Model intersection property). *Let $P = (R, D, T)$ be a well-defined fuzzy logic program and I and J be interpretations. Then*

$$I \Vdash P \text{ and } J \Vdash P \text{ implies } I \sqcap J \Vdash P.$$

Definition 7. *Let P be a well-defined fuzzy logic program. The* least model *of P is defined as* $\mathsf{lm}(P) := \bigsqcap_{I \Vdash P} I$.

3.2 Least fixpoint semantics

Definition 8 (T_P operator). *Let $P = (R, D, T)$ be a fuzzy logic program, I an interpretation, and recall that $\mathsf{ground}(R)$ denotes the set of all ground instances of clauses in R. The operator T_P is defined as follows:*

$$T_P(I) := \left\{ A \mapsto zv \;\middle|\; \begin{array}{l} A \xleftarrow{c,F_c}_F B_1, \ldots, B_n \in \mathsf{ground}(R), \\ I(B_i) = z_i v_i \succ \bot \text{ for all } i, \\ z = z_1 \circ \cdots \circ z_n, v = \hat{F}_c(c, \hat{F}(v_1, \ldots, v_n)) \end{array} \right\}$$

$$\sqcup \left\{ A \mapsto \blacktriangle\delta_j \;\middle|\; \begin{array}{l} \texttt{default}(p(X_1, \ldots, X_n)) \qquad = \\ [\delta_1 \text{ if } \varphi_1, \ldots, \delta_m \text{ if } \varphi_m] \in D, \\ A = p(t_1, \ldots, t_n), \text{ there exists a } 1 \le j \le m \\ \text{such that } \varphi_j[X_1/t_1, \ldots, X_n/t_n] \text{ holds} \end{array} \right\}$$

Note that if for a ground atom A both a program clause with body atoms from I's domain and a default value declaration exist, the truth value that comes from the clause is preferred.

Proposition 9 **(T_P is continuous).** *Let P be a well-defined fuzzy logic program and $I_0 \sqsubseteq I_1 \sqsubseteq \ldots$ a countably infinite increasing sequence of interpretations. Then*

$$T_P\left(\bigsqcup_{n=0}^{\infty} I_n\right) = \bigsqcup_{n=0}^{\infty} T_P(I_n).$$

We remark that continuity of an operator implies monotonicity, i.e. for two interpretations I, J with $I \sqsubseteq J$, we have $T_P(I) \sqsubseteq T_P(J)$.

We now recall the definition of ordinal powers of an operator. Let T be an operator and α be an ordinal number. The *ordinal power α of T* is defined as follows:

$$T{\uparrow}\alpha := \begin{cases} T(T{\uparrow}\alpha - 1) & \text{if } \alpha \text{ is a successor ordinal} \\ \bigsqcup_{\alpha' < \alpha} T{\uparrow}\alpha' & \text{if } \alpha \text{ is a limit ordinal} \end{cases}$$

Theorem 10. *Let P be a well-defined fuzzy logic program. Then the least fixpoint of T_P exists and is equal to $T_P{\uparrow}\omega$.*

Proof. *This follows from the facts that $(\mathcal{I}_P, \sqsubseteq)$ forms a complete lattice, T_P is continuous (Theorem 9), and the Knaster-Tarski fixpoint theorem [9].* □

Since the least fixpoint always exists (and always can be reached in countably many iterations), we can define a semantics based on it.

Definition 11. *Let P be a well-defined fuzzy logic program. Then the* least fixpoint semantics of P *is defined as* $\mathsf{lfp}(P) = T_P \uparrow \omega(\bot)$. *Here, $\bot$ denotes the interpretation mapping everything to $\bot$ (thus being the least element of the lattice $(\mathcal{I}_P, \sqsubseteq)$).*

Lemma 12. *Let P be a well-defined fuzzy logic program. Then*

$$I \Vdash P \quad \textit{iff} \quad T_P(I) \sqsubseteq I.$$

Theorem 13. *For a well-defined fuzzy logic program P, we have* $\mathsf{lm}(P) = \mathsf{lfp}(P)$.

Proof.

$$\begin{aligned} \mathsf{lm}(P) &= \bigsqcap_{I \Vdash P} I && \textit{(by definition)} \\ &= \bigsqcap_{T_P(I) \sqsubseteq I} I && \textit{(by Lemma 12)} \\ &= T_P{\uparrow}\omega(\bot) && \textit{(by the Kleene fixpoint theorem)} \\ &= \mathsf{lfp}(P) && \textit{(by definition)} \end{aligned}$$

□

4 Conclusions

We presented the RFuzzy framework – a logic programming tool that combines multi-adjoint semantics with some additional expressive characteristics as default values, types, simple data representation, and constructive answers.

A least model semantics and a least fixpoint semantics for RFuzzy have been defined and shown to be equivalent.

References

1. S. Guadarrama, S. Munoz-Hernandez, and C. Vaucheret. Fuzzy Prolog: A new approach using soft constraints propagation. *Fuzzy Sets and Systems, FSS*, 144(1):127–150, 2004. ISSN 0165-0114.
2. R. C. T. Lee. Fuzzy logic and the resolution principle. *Journal of the Association for Computing Machinery*, 19(1):119–129, 1972.
3. J. Medina, M. Ojeda-Aciego, and P. Votjas. A completeness theorem for multi-adjoint logic programming. In *International Fuzzy Systems Conference*, pages 1031–1034. IEEE, 2001.
4. J. Medina, M. Ojeda-Aciego, and P. Votjas. Multi-adjoint logic programming with continous semantics. In *LPNMR*, volume 2173 of *LNCS*, pages 351–364, Boston, MA (USA), 2001. Springer-Verlag.
5. Gines Moreno. Building a fuzzy transformation system. In *SOFSEM*, pages 409–418, 2006.
6. S. Munoz-Hernandez, C. Vaucheret, and S. Guadarrama. Combining crisp and fuzzy logic in a prolog compiler. In J. J. Moreno-Navarro and J. Mariño, editors, *Joint Conference on Declarative Programming: APPIA-GULP-PRODE 2002*, pages 23–38, Madrid, Spain, September 2002.
7. H. T. Nguyen and E. A. Walker. *A first Course in Fuzzy Logic.* Chapman & Hall/Crc, 2000.
8. Z. Shen, L. Ding, and M. Mukaidono. Fuzzy resolution principle. In *Proc. of 18th International Symposium on Multiple-valued Logic*, volume 5, 1989.
9. A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
10. C. Vaucheret, S. Guadarrama, and S. Munoz-Hernandez. Fuzzy prolog: A simple general implementation using clp(r). In P.J. Stuckey, editor, *International Conference in Logic Programming, ICLP 2002*, number 2401 in LNCS, page 469, Copenhagen, Denmark, July/August 2002. Springer-Verlag.