A Similarity-Based Resolution Rule

Francesca Arcelli Fontana,[1,*] Ferrante Formato[2]
[1] *University of Milano-Bicocca, via Bicocca degli Arcimboldi 8, 20126 Milano, Italy*
[2] *CRMPA, University of Salerno, 84084 Fisciano (SA), Italy*

We propose an extension of the resolution rule as the core of a logic programming language based on similarity. Starting from a fuzzy unification algorithm described in Ref. 2 and then extended in Ref. 10, we introduce a fuzzy resolution rule, based on an extended most general unifier supplied by the extended unification algorithm. In our approach, unification fades into a *unification degree* because of a similarity introduced in a first-order language. Intuitively, the unification degree of a set of first-order terms is the cost one has to pay to consider these terms as equal. For this reason, our extension of the resolution is more structured than its classic counterpart; that is, when the empty clause is reached, in addition to a computed answer, a set of conditions is also determined. We give both the operational and fixed-point semantics of our extended logic programming language, and we prove their equivalence. © 2002 Wiley Periodicals, Inc.

1. INTRODUCTION

In the field of databases and information retrieval systems there is an ever-increasing demand for systems able to deal with flexible queries and answers. Deductive databases, in particular, should cope with approximate inferences and more flexible information retrieval methods. In this article, we extend the classic resolution, and consequently, the classic refutation, into a more structured process. This extension is particularly significant when, in the classic case, a failure occurs. Such a technique can be usefully exploited for query evaluation in deductive databases. In our extended resolution, the classic unifier is replaced with a substitution obtained by a similarity-based extension of a unification algorithm, in which a unifier is associated with a set of conditions called *clouds*. Intuitively, a cloud is a set of elements that can be considered similar to a certain degree.

We derive an extension of the resolution rule as a process that, starting with a goal formula, as usual, collects a set of clouds stemming from each step of unification, and determines the degree to which this set is transformed into a set of singletons. This number is the degree of derivation of the goal formula. Through extended unification and resolution, we developed an extended logic programming language called Likelog.[3] The extended resolution is used to provide the operational semantics

* Author to whom all correspondence should be addressed; e-mail: arcelli@disco.unimib.it.

INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, VOL. 17, 853–872 (2002)
© 2002 Wiley Periodicals, Inc. Published online in Wiley InterScience (www.interscience.wiley.com). • DOI: 10.002/int.10067

of Likelog, and the extension principle described in Ref. 11 is used to define an extension of the least fixed-point operator to give the fixed-point semantics. The proof of the equivalence of the two semantics is also provided in this article.

This article is organized into the following sections. In Section 2, we define the basic concepts, that is, similarity, clouds, and closure operators. In Section 3, we describe our similarity-based unification algorithm. In Section 4, we introduce an extension of the classic resolution rule and prove how it can be used to provide the operational semantics. In Section 5, we provide a fixed-point semantics through a similarity-based extension of the least Herbrand model. Finally, we outline how it is possible to extend the question-and-answer process to a deductive database by introducing similarity.

1.1. Related Works

Several fuzzy extensions of the resolution rule of Ref. 17 used in classic logic programming have been proposed in the fuzzy context. The first work on fuzzy resolution for a set of ground clauses was provided by Lee[13]; other extensions of resolution in fuzzy logic can be found in subsequent works, as for example Ref. 4, in which resolution has been generalized to clauses involving universally quantified variables, and the problem of searching for the most informative proof has been pointed out. In Refs. 16 and 18, a fuzzy resolution principle is defined at first for propositional logic and then for first-order logic, with the proof of its completeness, and the description of a fuzzy Prolog system based on it. An extension of the resolution principle to possibilistic logic, in which clauses are weighted with a degree that represents a possibilistic measure is described in Ref. 8. Resolution in the context of possibilistic logic with fuzzy constants is discussed in Ref. 9, and in Refs. 5 and 6, fuzzy resolution has been defined in the context of evidential logic. In Ref. 20, a λ-resolution method is proposed and its completeness proven in the context of *operator fuzzy logic*. Virtanen in Ref. 19 shows how SLD resolution used in logic programming can be modified and used for linguistic logic programming, in which variables can take fuzzy terms. This kind of resolution is based on the notion of fuzzy equality. In Ref. 21, a refutation method based on *tree resolution* is proposed in which truth values are assigned to the program clauses on the basis of Lukasiewicz implication.

In most cases, the fuzzy extension of the resolution rule does not stem from a fuzzy extension of the unification algorithm, and classic unification is used. In some cases, it is necessary to use fuzzy unification methods, such as the one we describe here.

2. SIMILARITY, CLOUDS, AND CLOSURE OPERATORS

The theoretical foundations of the fuzzy resolution that we propose are based on the notion of similarity. Similarity is a many-valued extension of the classic notion of equivalence, and it is strictly connected to the notion of a fuzzy subset: given a set S, a *fuzzy subset of* S is a map $s : S \to [0, 1]$. We denote the class of fuzzy subsets of S as $\mathcal{F}(S)$.

Let S be a non-empty set; we denote its power set by $\mathcal{P}(S)$. Given a fuzzy subset s and $\lambda \in [0, 1]$, we call the set $\mathcal{C}(s, \lambda) = \{x \in S \mid s(x) \geq \lambda\}$ the *closed cut* of level λ. A family of subsets, $(C_\lambda)_{\lambda \in [0,1]}$, of S is called a *chain* if for any $\mu, \lambda \in [0, 1]$, the following properties are satisfied:

(1) $C_0 = S$
(2) $\lambda \leq \mu \Rightarrow C_\lambda \supseteq C_\mu$

A chain of sets is called *continuous* if:

(3) $\bigcap_{\lambda \leq \mu} C_\lambda = C_\mu$

DEFINITION 1. *A* similarity *on a set S is a fuzzy subset $\mathcal{R}$ of $S \times S$ such that, for any $x, y, z \in S$, the following properties hold*:

(1) $\mathcal{R}(x, x) = 1$ (reflexivity)
(2) $\mathcal{R}(x, y) = \mathcal{R}(y, x)$ (symmetry)
(3) $\mathcal{R}(x, z) \geq \mathcal{R}(x, y) \wedge \mathcal{R}(y, z)$ (transitivity)

The symbols $\wedge$ and $\vee$ correspond to the least upper bound (lub) and the greatest lower bound (glb) operators, respectively. Definition 1 can be generalized by replacing the minimum with any t-norm. Note that, in the case of the minimum, we must pay attention to the context in which we apply similarity. To show this, suppose that S is the set of integers, and $\mathcal{R}$ is a similarity such that $\mathcal{R}(x, x+1) \geq 0.7$, for any $x \in S$. Then $\mathcal{R}(x, y) \geq 0.7$ for any x, y in S, and this could not be satisfactory. Therefore, the use of the minimum is strictly dependent on the context, and other kinds of t-norms must be considered if we want to avoid this fact.

Given a similarity $\mathcal{R}$, we can associate to every non-empty set X of S a number $\mu(X)$ expressing the extent to which all the elements in X are similar:

$$\mu(X) = \begin{cases} \bigwedge\limits_{x,x' \in X} \mathcal{R}(x, x') & \text{if } X \neq \emptyset \\ 1 & \text{otherwise} \end{cases}$$

The number $\mu(X)$ is called the *co-diameter of* X. We call clouds the finite subsets of S, and therefore we say that $\mu(X)$ is the co-diameter of the cloud X. Intuitively, a cloud is a set of elements that can be considered as similar to one another. For example, consider the set $S = \{$*rectangle*, *square*, *rhombus*$\}$, in which *rectangle*, *square*, and *rhombus* indicate the corresponding well-known figures in the Euclidean plane. We set $\mathcal{R}$(rectangle, square) $= 0.7$, and $\mathcal{R}$(square, rhombus) $= \mathcal{R}$(rhombus, rectangle) $= 0.4$. Therefore, the cloud $\{$rectangle, square$\}$ indicates a set of plane figures that are considered similar one another.

Another interesting way to consider clouds is in the framework of games. Consider a two-player game in which a player P1 has a set of possible questions $\{q1, q2, q3\}$ and a player P2 has a set of possible answers $\{a1, a2, a3, a4\}$. Suppose that the two sets are made up of elements of the same nature, and that a similarity, $\mathcal{R}$, is defined on them. Then, for any snapshot of the game, a cloud $\{a_i, q_j\}$ is determined, and it can be considered as the conditions of the agreement of the two players, and the co-diameters of the cloud express such a degree.

A system of clouds is a finite set of clouds. Given a system of clouds $Z = \{X_1, \ldots, X_n\}$, we call the co-diameter of Z the number:

$$\xi(Z) = \bigwedge_{i=1}^{n} \mu(X_i)$$

We use the term co-diameter because the function $\mathcal{R}'(x, y) = 1 - \mathcal{R}(x, y)$ is a distance for which the related diameter is given by $1 - \mu(X) = \vee_{x,y \in X} \mathcal{R}'(x, y)$. The following properties of μ are given:

PROPOSITION 1. *Let $\{X_i\}_{i=1\ldots n}$ be a finite sequence of clouds such that, for any $i = 1 \ldots n-1$, $X_i \cap X_{i+1} \neq \emptyset$:*

$$\mu\left(\bigcup_{i=1}^{n} X_i\right) = \bigwedge_{i=1}^{n} \mu(X_i)$$

Moreover, assume that $X \neq \emptyset$ and $c \in X$. Then:

$$\mu(X) = \bigwedge_{x \in X} \mathcal{R}(x, c)$$

Proof. Clearly, $\bigwedge_{x \in X} \mathcal{R}(x, c) \geq \mu(X)$. Moreover, for any $x, y \in X$:

$$\mathcal{R}(x, y) \geq \mathcal{R}(x, c) \wedge \mathcal{R}(c, y) \geq \bigwedge_{x \in X} \mathcal{R}(x, c)$$

Therefore:

$$\mu(X) \geq \bigwedge_{x \in X} \mathcal{R}(x, c)$$

■

Given two clouds M and M', we say that M and M' *overlap* if $M \cap M' \neq \emptyset$.

DEFINITION 2. *We say that a fuzzy subset s is extensional with respect to a similarity $\mathcal{R}$ provided that, for any $x, y \in S$:*

$$s(x) \wedge \mathcal{R}(x, y) \leq s(y)$$

Definition 2 is a many-valued version of the claim "if x satisfies the property P and x is equal to y, then y satisfies property P, too."

2.1. An Operator over Systems of Clouds

We now introduce a generalization of an operator described in the unification algorithm of Ref. 15. We say that a system of clouds Z is *compact* if its elements are pairwise disjoint, that is, they do not overlap. We define an operator to obtain a compact system of clouds.

DEFINITION 3. *Let $Z = \{M_1, \ldots, M_n\}$ be a system of clouds. The following is an operator on the clouds of Z*:

$$T_Z(M) = \bigcup_{M' \cap M \neq \emptyset, M' \in Z} M'$$

The extension of T_Z to a system of clouds is as follows:

$$T(Z) = \{T_Z(M) \mid M \in Z\}$$

The operator T does not change the co-diameter of a system of clouds and can be iterated in the usual manner, by setting $T^0(Z) = Z$ and $T^{n+1}(Z) = T(T^n(Z))$ for any integer $n \geq 1$.

DEFINITION 4. *Let Z be a system of clouds. We define the* degree, $\varphi(Z)$, *of the system Z as follows*:

$$\varphi(Z) = \min_n \{T^n(Z) = T^{n+1}(Z)\}$$

Since Z is a system of finite clouds, the degree of Z is a finite non-negative integer. Obviously, if Z is a compact system of clouds, $\varphi(Z) = 0$ and Z is a fixed-point of T.

DEFINITION 5. *Let Z be a system of clouds. The operator* Compact *is defined as follows*:

$$Compact(Z) = T^{\varphi(Z)}(Z)$$

By the definition of $\varphi(Z)$, the operator *Compact* transforms a system of clouds into a compact system. With respect to its classic counterpart, our *Compact* operator acts on systems of clouds and involves constants, too.

The following proposition describes some important properties of *Compact*.

PROPOSITION 2. *Let Z be a system of clouds. Then*:

(1) $\xi(Compact(Z)) = \xi(Z)$
(2) $Compact(Compact(Z)) = Compact(Z)$

Proof. Property (1) is immediate by the definition of the co-diameter and Proposition 1.

Property (2) is immediate, because $Compact(Z)$ is compact. ■

Properties above will be used in the definition of extended resolution.

2.2. An Extension Principle for Fuzzy Closure Operators

As with the classic case, we use fuzzy closure operators to provide a fixed-point semantics for our extended logic programming language.

A *closure operator* on S is any map $F: \mathcal{P}(S) \to \mathcal{P}(S)$ such that, for any $X, Y \in \mathcal{P}(S)$, $F(X) \supseteq X$; $X \supseteq Y \Rightarrow F(X) \supseteq F(Y)$; $F(F(X)) = F(X)$.

A family $(J_\lambda)_{\lambda\in[0,1]}$ of closure operators on S is called a *chain* (or a *continuous chain*), if for every $X \subseteq S$, $(J_\lambda(X))_{\lambda\in[0,1]}$ is a chain (or a continuous chain) of subsets.

A fuzzy extension of the notion of closure operator is defined accordingly.

DEFINITION 6. *A fuzzy operator on S is any map $J: \mathcal{F}(S) \to \mathcal{F}(S)$. A fuzzy closure operator on S is a fuzzy operator on S such that, for any $s, s' \in \mathcal{F}(S)$:*

(1) $J(s) \supseteq s$ (*inclusion*)
(2) $J(s') \subseteq J(s)$ *if* $s' \subseteq s$ (*monotony*)
(3) $J(J(s)) = J(s)$ (*idempotence*)

There is a close connection between classic closure operators and their fuzzy counterparts. In particular, the following theorem proven in Ref. 11 exploits an extension principle that allows the straightforward extension of any classic operator.

THEOREM 1. *Let $(J_\lambda)_{\lambda\in[0,1]}$ be a chain of closure operators on S and let J be the fuzzy operator on S defined by setting, for any $s \in \mathcal{F}(S)$ and $y \in S$:*

$$J(s)(y) = \sup\{\lambda \in [0, 1] \mid y \in J_\lambda(C(s, \lambda))\}$$

Then J is a fuzzy closure operator.

A fuzzy closure operator that is obtained by applying the extension principle to a chain of classic closure operators is called *stratified*.

If $(\mathcal{R}_\lambda)_{\lambda\in[0,1]}$ is a family of equivalence relations on S such that $\mathcal{R}_0 = S$ and $\mathcal{R}_\lambda \subseteq \mathcal{R}_{\lambda'}$ if $\lambda' \leq \lambda$, then for any $\lambda \in [0, 1]$, the operator $H_\lambda(X) = \{y \in S \mid y\ \mathcal{R}_\lambda\ x$ for some $x \in X\}$ is a closure operator on S, and for every $X \subseteq S$, $(H_\lambda(X))_{\lambda\in[0,1]}$ is a chain of subsets. Therefore, H_λ is a chain of closure operators on S and the following fuzzy operator H on $\mathcal{F}(S)$ defined by setting, for any $s \in \mathcal{F}(S)$, is a stratified fuzzy closure operator on S:

$$H(s)(y) = \sup\{\lambda \in [0, 1] \mid y \in H_\lambda(\mathcal{C}(s, \lambda))\}$$

3. A SIMILARITY-BASED UNIFICATION ALGORITHM

Our investigation follows two paths: we first define an extended unification theory that is based on the fuzzification of the meta-logical notions of its classic analog, then we extend the unification algorithm of Ref. 15 and show that this algorithm is correct, that is, it produces an extended most general unifier as output.

We use a similarity to extend the first-order unification and resolution. To do that, let $\mathcal{L}$ be a classic first-order language; we also denote:

- $\mathcal{V}$ as the *set of variables*
- $\mathcal{C}$ as the *set of constants*
- P as the *set of predicate symbols*
- $\mathcal{T}_{\mathcal{V},\mathcal{C}}$ as the *set of terms,* that is, the set $\mathcal{V} \cup \mathcal{C}$

We expand the language $\mathcal{L}$ into the language $\mathcal{L}'$, for which the set of constants is the set $\mathcal{C}' = \mathcal{P}(\mathcal{C}) - \{\emptyset\}$, which is called the *set of extended constants*.

We denote the *set of extended terms,* or *e-terms,* by $\mathcal{T}_{\mathcal{V},\mathcal{C}'}$, that is, the set $\mathcal{V} \cup \mathcal{C}'$. An *extended substitution* (for short, an *e-substitution*) is a map $\theta : \mathcal{V} \to \mathcal{T}_{\mathcal{V},\mathcal{C}'}$, and we denote the set of e-substitutions by Θ. An *atomic extended formula,* or *atomic e-formula,* is an expression of the type $p(t_1, \ldots, t_n)$, where $p \in \mathrm{P}$ and $t_i \in \mathcal{T}_{\mathcal{V},\mathcal{C}'}$ for any $i = 1, \ldots, n$. Finally, an *extended equation,* or *e-equation,* is an expression of the type $A = B$, where A and B are atomic e-formulae with the same arity.

We identify any constant c in $\mathcal{C}$ with the singleton $\{c\}$ in $\mathcal{C}'$. Given two similarities over P and $\mathcal{C}$, denoted respectively as $\mathcal{R}_P$ and $\mathcal{R}_\mathcal{C}$, we define a similarity $\mathcal{R}$ on $P \cup \mathcal{C} \cup \mathcal{V}$ in the following way:

$$\mathcal{R} = \mathcal{R}_P \cup \mathcal{R}_\mathcal{C} \cup \mathcal{R}_\mathcal{V}$$

where $\mathcal{R}_\mathcal{V}$ is a similarity on $\mathcal{V}$ defined as follows:

$$\mathcal{R}_\mathcal{V}(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$$

Given a system of e-equations, S, and an e-substitution, θ, we define the *unification degree of* S with respect to θ by the expression:

$$\nu(S, \theta) = \bigwedge_{e=e' \in S} r(\theta(e) = \theta(e'))$$

where, if $e = p(t_1, \ldots, t_n)$ and $e' = q(t'_1, \ldots, t'_n)$,

$$r(\theta(p(t_1, \ldots, t_n)) = \theta(q(t'_1, \ldots, t'_n))) = \bigwedge_{i=1,\ldots,n} \mathcal{R}(p, q) \wedge \mathcal{R}(\theta(t_i), \theta(t'_i))$$

DEFINITION 7. *Given a system of e-equations, S, we define the unification degree of S by the following expression*:

$$U(S) = \bigvee_{\theta \in \Theta} \nu(S, \theta)$$

Given a system of clouds, Z, in $\mathcal{T}_{\mathcal{V},\mathcal{C}}$ of the kind $\{M_1, \ldots, M_n\}$, we write a generic cloud M in Z in the form $X \sqcup D$, where $X = M \cap \mathcal{V}$ and $D = M \cap \mathcal{C}$. Analogously, given an e-substitution, θ, by $\theta(M)$ we denote the cloud $\bigcup_{x \in X} \theta(X) \sqcup D$, identifying a variable y with its singleton $\{y\}$ where necessary.

An *extended unifier* (e-unifier) of a system of e-equations, S, is an e-substitution, γ, such that $\nu(S, \gamma) = U(S)$.

To any compact system of clouds, Z, in $\mathcal{T}_{\mathcal{V},\mathcal{C}}$, we associate an e-substitution, θ_Z, defined as follows:

$\theta_Z(x) = x$ if $x \notin M_1 \cup \cdots \cup M_n$

$\theta_Z(x) = M_i \cap \mathcal{C}$, provided that $x \in M_i$ and $M_i \cap \mathcal{C} \neq \emptyset$

$\theta_Z(x) = x_{j(i)}$, provided that $x \in M_i$, $M_i \cap \mathcal{C} = \emptyset$ and $x_{j(i)}$ is a variable in M_i

As an example, we can assume that $x_{j(i)}$ is the first variable in the sequence $x_1, x_2, \ldots$, belonging to M_i, provided that an order relation exists on $\mathcal{V}$.

To define our unification algorithm, we introduce some other operators and functions.

Given a cloud $X \sqcup D$, we set $Ground(X \sqcup D) = D$, and given a set of clouds Z, we set:

$$Ground(Z) = \{Ground(M) \mid M \in Z\}$$

Let t and t' be two clouds; then $t \dot{\sqcup} t'$ is a cloud defined as follows:

$$t \dot{\sqcup} t' = \begin{cases} t \cup t' & \text{if } t \quad \text{and} \quad t' \in \mathcal{C}' \\ \{t\} \cup t' & \text{if } t \in \mathcal{V}, t' \in \mathcal{C}' \\ \{t'\} \cup t & \text{if } t \in \mathcal{C}', t' \in \mathcal{V} \\ \{t, t'\} & \text{if } t, t' \in \mathcal{V} \end{cases}$$

Now we consider two functions *Trans_t* and *Trans_p* transforming a system of e-equations into systems of clouds in $\mathcal{C} \cup \mathcal{V}$ and P, respectively.

We use the following functions:

$$Split\_t(p(t_1, \ldots, t_m) = q(t'_1, \ldots, t'_m)) = \{t_1 \dot{\sqcup} t'_1, \ldots, t_m \dot{\sqcup} t'_m\}$$
$$Split\_p(p(t_1, \ldots, t_m) = q(t'_1, \ldots, t'_m)) = \{p, q\}$$

If $S = \bigcup_{i=1}^{n} \{e_i = e'_i\}$, we set:

$$Trans\_t(S) = \bigcup_{i=1 \ldots n} Split\_t(e_i = e'_i)$$
$$Trans\_p(S) = \bigcup_{i=1 \ldots n} Split\_p(e_i = e'_i)$$

We associate an e-substitution to any compact system Z by setting $Assoc\_sub(Z) = \theta_Z$.

For a given system of e-equations S, let:

$$\theta_S = Assoc\_sub(Compact(Trans\_t(S)))$$

We now give a set of definitions and results used to prove the fundamental theorem from which our extended unification algorithm has been derived.

DEFINITION 8. *Given a system of clouds, Z, the unification degree of Z with respect to an e-substitution, θ, is defined by $\nu(Z, \theta) = \xi(\theta(Z))$. We define the unification degree of Z by the following expression*:

$$U(Z) = \bigvee_{\theta \in \Theta} \nu(Z, \theta)$$

If $U(Z) \neq 0$, an e-unifier *of Z is an e-substitution, θ, such that $\nu(Z, \theta) = U(Z)$.*

The proof of the following proposition is immediate.

PROPOSITION 3. *Let Z be a compact system of clouds such that $U(Z) \neq 0$. Then $Assoc\_sub(Z)$ is an e-unifier of Z, and we have*:

$$U(Z) = \xi(Ground(Z))$$

Proof. For any e-substitution, θ:

$$\xi(\theta(Z)) = \bigwedge_{X \sqcup D \in Z} \mu(\theta(X \sqcup D)) \leq \bigwedge_{X \sqcup D \in Z} \mu(D) = \xi(Ground(Z))$$

Since $\xi(Assoc\_sub(Z)(Z)) = \xi(Ground(Z))$, the thesis follows. ∎

PROPOSITION 4. *Let Z be a system of clouds. Then, for any e-substitution, θ, we have*:

$$\xi(\theta(Compact(Z))) = \xi(\theta(Z))$$

Proof. We first prove that

$$\xi(\theta(T(Z))) = \xi(\theta(Z)) \tag{1}$$

We begin by proving that, for $i = 1, \ldots, n$,

$$\mu(\theta(T_Z(M_i))) \geq \xi(\theta(Z)) \tag{2}$$

Indeed, because from $M_i \cap M_j \neq \emptyset$ we have that $\theta(M_i) \cap \theta(M_j) \neq \emptyset$, by Proposition 2:

$$\begin{aligned}
\mu(\theta(T_Z(M_i))) &= \mu\left(\theta\left(\bigcup_{M_j \cap M_i \neq \emptyset} M_j\right)\right) \\
&= \mu\left(\bigcup_{M_j \cap M_i \neq \emptyset} \theta(M_j)\right) = \bigwedge_{M_j \cap M_i \neq \emptyset} \mu(\theta(M_j)) \\
&\geq \bigwedge_{i=1}^{n} \mu(\theta(M_i)) = \xi(\theta(Z))
\end{aligned}$$

That proves the inequality in Equation 2.

From Equation 2, it follows that:

$$\xi(\theta(T(Z))) = \bigwedge_{i=1}^{n} \mu(\theta(T_Z(M_i))) \geq \xi(\theta(Z))$$

On the other hand, because for any $i = 1, \ldots, n$, $M_i \subseteq T_Z(M_i)$,

$$\xi(\theta(Z)) = \bigwedge_{i=1}^{n} \mu(\theta(M_i)) \geq \bigwedge_{i=1}^{n} \mu(\theta(T_Z(M_i)) = \xi(\theta(T(Z)))$$

This completes the proof of Equation 1.

We proceed by induction on the T-degree $\Phi(Z)$ of Z.

- If $\Phi(Z) = 0$, then Z is compact and therefore the thesis follows.
- If $\Phi(Z) = n$, then $\Phi(T(Z)) = n - 1$.

Therefore, by induction hypothesis:

$$\xi(\theta(Compact(T(Z)))) = \xi(\theta(T(Z)))$$

Since Z has T-degree n,

$$\begin{aligned} Compact(Z) &= T^n(Z) \\ &= T^{n-1}(T(Z)) = Compact(T(Z)) \end{aligned}$$

Then, by Equation 1, $\xi(\theta(Compact(Z))) = \xi(\theta(Z))$. ∎

Now we are ready to prove that the unification degree of a system of clouds or e-equations is an invariant with respect to the transformations *Compact Trans_t* and *Trans_p*, as stated by the following theorem.

THEOREM 2. *Let S be a system of e-equations. Then, for any extended substitution, θ, we have*:

(1) $\nu(S, \theta) = \nu(Trans\_t(S), \theta) \wedge \xi(Trans\_p(S))$
and therefore:
(2) $\nu(S, \theta) = \nu(Compact(Trans\_t(S)), \theta) \wedge \xi(Compact(Trans\_p(S)))$.

Proof. (1) Observe that, if $e = p(t_1, \ldots, t_n)$ and $e' = q(t'_1, \ldots, t'_n)$ then, since $Split\_t(\theta(e_i) = \theta(e'_i)) = \theta(Split\_t(e_i = e'_i))$, for any $i = 1, \ldots, n$, we have:

$$\begin{aligned} \overline{eq}(\theta(e), \theta(e')) &= \left(\bigwedge_{i=1}^{n} \mu(\theta(t_i) \dot{\sqcup} \theta(t'_i)) \right) \wedge eq(p, q) \\ &= \left(\bigwedge_{i=1}^{n} \mu(\theta(t_i \dot{\sqcup} t'_i)) \right) \wedge eq(p, q) \\ &= \xi(\theta(Split\_t(e = e'))) \wedge \xi(Split\_p(e = e')) \end{aligned}$$

Therefore:

$$\begin{aligned} \nu(S, \theta) &= \bigwedge_{e=e' \in S} \overline{eq}(\theta(e), \theta(e')) \\ &= \bigwedge_{e=e' \in S} \xi(\theta(Split\_t(e = e'))) \wedge \bigwedge_{e=e' \in S} \xi(Split\_p(e = e')) \\ &= \xi\left(\theta\left(\bigcup_{e=e'} Split\_t(e = e') \right) \right) \wedge \xi\left(\bigcup_{e=e'} Split\_p(e = e') \right) \\ &= \xi(\theta(Trans\_t(S))) \wedge \xi(Trans\_p(S)) \end{aligned}$$

(2) Follows immediately from (1) and Proposition 4. ∎

We can now state our main result, used to derived the similarity-based unification algorithm.

THEOREM 3. *Let S be a system of e-equations such that $U(S) > 0$. Then*:

(1) $U(S) = \xi(Ground(Compact(Trans\_t(S)))) \wedge \xi(Trans\_p(S))$
(2) θ_S *is an e-unifier for* S

Proof. An immediate consequence of Theorem 2 and Proposition 3. ■

Indeed, the above theorem assures that an extended unifier can be computed by our extended unification algorithm. In fact, by property (1), given a system S such that $U(S) > 0$, we can obtain an e-unifier of S by computing the value of the function $Assoc\_sub(Compact(Trans\_t(S)))$.

In order to provide an extension of the most general unifier, we proceed as follows. Let M and M' be two clouds; we write $M \cong_\lambda M'$ if $\mu(M \cup M') \geq \lambda$. Given two e-substitutions θ and θ', we write $\theta \cong_\lambda \theta'$, provided that $\theta(x) \cong_\lambda \theta'(x)$ for any $x \in \mathcal{V}$. We have the following definition.

DEFINITION 9. *An e-substitution γ is called an extended most general unifier* (*e-mgu*) *for a system of e-equations, S, if γ is an e-unifier of S and for any e-unifier, θ, of S, $\theta \cong_{U(S)} \tau\gamma$ for some e-substitution τ.*

Note that $\cong_{U(S)}$ is an equivalence relation among e-unifiers of S.

It is proven that, if $U(S) > 0$, then θ_S is an e-mgu for S.[10] As a consequence, an e-mgu can be computed by the extended unification algorithm described above.

4. A SIMILARITY-BASED RESOLUTION RULE

Our definition of extended resolution starts from a straightforward extension of the classic SLD resolution for definite Horn clauses, as in Ref. 1. A *definite* (*logic*) *program* P is a finite set of definite Horn clauses. An *extended resolvent* (*e-resolvent*) is an e-formula of the kind $A_1 \wedge \cdots \wedge A_k$, (for short, $A_1, \ldots, A_k$) in which each A_i is an atomic e-formula. If $k = 0$, we say that the e-resolvent is *empty*. Given an e-substitution, θ, we indicate the formula $\theta(A_1) \wedge \cdots \wedge \theta(A_k)$ by $\theta(A_1 \wedge \cdots \wedge A_k)$.

As usual, a *variant* of a clause C in a definite program P is a renaming of the free variables occurring in C. In the following definition we introduce our similarity-based resolution rule.

DEFINITION 10. *Let P be a definite program, $N = A_1 \wedge \cdots \wedge A_k$ an e-resolvent, and $C = A \leftarrow B_1, \ldots, B_n$ a variant of a clause in P. The e-resolvent $N' = \theta(A_1 \wedge \cdots \wedge A_{i-1} \wedge B_1 \wedge \cdots B_n \wedge A_{i+1} \wedge \cdots \wedge A_k)$ is called an e-resolvent of N and C with e-mgu θ and a set of conditions Cond, provided that, for some $i \in \{1, \ldots, n\}$, the e-formula A_i unifies with the atomic e-formula in the head of C according to the*

similarity-based unification (*described in Section* 3), *yielding an e-mgu* θ *and a set of conditions Cond expressed as*:

$$Cond = Ground(Compact(Trans\_t(S)) \cup Compact(Trans\_p(S))$$

where $S = \{A = Head(C)\}$.

We are now ready to give the following definition of the extended SLD derivation.

DEFINITION 11. *Let P be a definite program and let N be an e-resolvent. An extended SLD derivation of* $P \cup \{N\}$, *or for short*, *e-SLD derivation*, *is a maximal sequence* N_i *of e-resolvents*, *a sequence* C_i *of variants of clauses*, *a sequence* θ_i *of e-substitutions*, *a sequence* S_i *of systems of e-equations*, *and a sequence* $\mathcal{C}_i$ *of systems of clouds*, *such that*, *for any* i:

- $N_0 = N$.
- N_{i+1} *is an e-resolvent of* N_i *and* C_i, *with* θ_i *as e-mgu*, *and* $\mathcal{C}_i$ *is a set of conditions.*
- $S_i = \{A_i = head(C_i)\}$, *where* A_i *is an atomic e-formula in* N_i *and* $head(C_i)$ *is the atomic formula in the head of the clause* C_i.
- $\mathcal{C}_i = Ground(Compact(Trans\_t(S_i)) \cup Compact(Trans\_p(S_i))$.
- θ_i *is an e-mgu of the system of* S_i *obtained by setting* $\theta_i = \theta_{S_i}$.
- C_i *has no variable in common with* $N_0, C_0, \ldots, C_{i-1}$.

The clauses C_i are called *input clauses,* as usual.

If, for some integer $\bar{n}$, $N_{\bar{n}}$ is empty, then the e-SLD derivation ends and is called an e-*SLD refutation.* In this case, the set $\bigcup_{i=1,\ldots,\bar{n}} \mathcal{C}_i$ is called the *conditions set* of the e-SLD refutation. The restriction of the e-substitution $\theta_{\bar{n}} \circ \theta_{\bar{n}-1} \circ \ldots \theta_1$ to the variables in N is called the *e-computed answer substitution* for $P \cup \{N\}$.

If an e-SLD derivation is finite and is not an e-refutation, it is called a *failed* derivation.

DEFINITION 12. *Given an e-refutation* $r_{P,N}$ *of* $P \cup \{N\}$ *ending after* $\bar{n} + 1$ *resolution steps*, *we define the degree of the e-refutation of* $P \cup \{N\}$ *as* $h(r_{P,N})$, *and denote it as follows*:

$$h(r_{P,N}) = \xi\left(Compact\left(\bigcup_{i=0,\ldots,\bar{n}} \mathcal{C}_i\right)\right)$$

In the following, we give an example of an e-SLD refutation.

Example 1. Let P be the following program:

$$p(x) \leftarrow q(x), r(x)$$
$$q'(a)$$
$$r'(b)$$
$$k(a)$$

Let $N_0 = \{p(x)\}$; then we have the following e-SLD derivation of $P \cup \{N\}$:

$$\begin{aligned}
\mathbf{N}_0 &= p(x) \\
C_0 &= p(x_1) \leftarrow q(x_1), r(x_1) \\
\theta_0 &= \{x \to x_1\} \\
S_0 &= \{p(x) = p(x_1)\} \\
\mathcal{C}_0 &= \{\{p\}\} \\
\mathbf{N}_1 &= q(x_1), r(a) \\
C_1 &= q'(a) \\
\theta_1 &= \{x_1 \to a\} \\
S_1 &= \{q(x_1) = q'(a)\} \\
\mathcal{C}_1 &= \{\{q, q'\}\} \\
N_2 &= r(a) \\
C_2 &= r'(b) \\
\theta_2 &= \{\} \\
S_2 &= \{r(a) = r'(b)\} \\
\mathcal{C}_2 &= \{\{r, r'\}, \{a, b\}\} \\
N_3 &= []
\end{aligned}$$

In this case, the e-SLD resolution chain is an e-refutation, and the e-computed answer substitution of the e-refutation is:

$$\theta = \{x \to \{a, b\}\}$$

The degree of the e-refutation is:

$$\xi\left(Compact\left(\bigcup_{i=0,\ldots,4} \mathcal{C}_i\right)\right) = \xi(\{\{q, q'\}, \{r, r'\}, \{a, b\}\})$$

This means that the cost to turn the clouds $\{a, b\}$, $\{q, q'\}$, and $\{r, r'\}$ into a set of singletons, transforming the e-refutation into a classic refutation, is given by $\xi(\{\{q, q'\}, \{r, r'\}, \{a, b\}\})$.

Note that an e-resolvent, N, can have several distinct e-refutations with respect to the same definite program, P.

DEFINITION 13. *Let $\Pi(P, N)$ be the set of e-refutations of $P \cup \{N\}$. We say that an e-resolvent, N, is derivable from P with degree λ if*:

$$\lambda = \bigvee_{r \in \Pi(P,N)} h(r)$$

Recalling that a ground formula is a formula with no variables, we can define a fuzzy subset, $\mathcal{D}_P$, of atomic ground formulae by setting, for any atomic e-formula A:

$$\mathcal{D}_P(A) = \bigvee_{r \in \Pi(P,A)} h(r)$$

$\mathcal{D}_P$ can be seen as the *fuzzy subset of the logical consequences* of a definite logic program P. It can be viewed as the extension of the operational semantics of our logic programming system based on similarity. An e-refutation is called ground if all its resolvents are ground.

The following theorem is an immediate generalization of the classic case.

THEOREM 4. *Let G be a ground atomic formula. For any e-refutation r in $\Pi(P, G)$ there exists a ground e-refutation r' in $\Pi(P, G)$ such that $h(r) = h(r')$.*

Proof. Let $r = (N_i, \theta_i, S_i, \mathcal{C}_i)$ be an e-refutation of G, let θ be the e-computed answer of r, and let τ be an e-substitution defined as follows:

$$\tau(x) = \begin{cases} \theta(x) & \text{if } \theta(x) \text{ is an extended constant} \\ a \notin \mathcal{C}' & \text{otherwise} \end{cases}$$

We set r' in the following way:
$N'_i = \tau\theta(N_i)$
$S'_i = \tau\theta(S_i)$
$\mathcal{C}'_i = Ground(Comp(Trans\_t(S'_i))) \cup Ground(Comp(Trans\_p(S'_i)))$
$\theta_i = \epsilon$
It is easy to see that r' is an e-refutation of G with respect to P such that $h(r') = h(r)$. ∎

We extend the similarity $\mathcal{R}$ to the set of atomic e-formulae by setting, for any α, β:

$$\mathcal{R}(\alpha, \beta) = \begin{cases} \bigwedge_{i=1}^{n} \mathcal{R}(t_i, t'_i) \wedge \mathcal{R}(p, p') & \text{if } \alpha = p(t_1, \ldots, t_n) \\ \beta = p'(t'_1, \ldots, t'_n) & \\ 0 & \text{otherwise} \end{cases}$$

On account of the theorem above, we can prove the following lemma.

LEMMA 1. *Let G and G' be two ground atomic formulae; for any $r \in \Pi(P, G)$ there exists an e-refutation r' in $\Pi(P, G')$ such that*:

$$h(r') \geq \mathcal{R}(G, G') \wedge h(r)$$

Proof. Let r be an e-refutation of G. On account of Theorem 4, we can consider r as ground. Let $h(r) = \mathcal{R}(G, C) \wedge \xi(\mathcal{C})$, where C is the head of a ground instantiation of clause in P and $\mathcal{C}$ is a suitable set of conditions. By replacing the first resolvent

N_0 in r with G', we get a ground e-refutation r' such that:

$$\begin{aligned} h(r') &= \mathcal{R}(G', C) \wedge \xi(\mathcal{C}) \geq \mathcal{R}(G, G') \wedge \mathcal{R}(G', C) \wedge \xi(\mathcal{C}) \\ &= \mathcal{R}(G', G) \wedge h(r) \end{aligned}$$

Hence,

$$h(r') \geq R(G, G') \wedge h(r)$$ ■

Now we can prove the following property.

PROPOSITION 5. *$\mathcal{D}_P$ is extensional with respect to the similarity $\mathcal{R}$; that is, for any pair of atomic ground formulae α and β:*

$$\mathcal{D}_P(\alpha) \geq \mathcal{R}(\alpha, \beta) \wedge \mathcal{D}_P(\beta)$$

Proof. Let α and β be two atomic ground formulae. Let $r \in \Pi(P, \alpha)$. On account of Lemma 1, there is an e-refutation r' of β such that:

$$\mathcal{D}_P(\beta) \geq h(r') \geq h(r) \wedge \mathcal{R}(\alpha, \beta)$$

Therefore, considering the supremum of $h(r)$ over $\Pi(P, \alpha)$ we have:

$$\mathcal{D}_P(\beta) \geq \mathcal{D}_P(\alpha) \wedge \mathcal{R}(\alpha, \beta)$$ ■

The proposition above is the many-valued interpretation of the fact that, if a formula β is derivable from the program P and $\alpha = \beta$, then β is derivable from α, too.

5. SIMILARITY-BASED HERBRAND MODELS

Extended SLD resolution can be used as the operational semantics of a similarity-based logic programming language, and the fixed-point semantics is given by the extension of the least Herbrand model of a program. We prove that the extension of the least Herbrand model of a program P, obtained through the application of the extension principle given in Section 2, coincides with the fuzzy set of ground facts that are derived by extended refutations.

Recall that the *Herbrand universe* and the *Herbrand base* of a program, P, denoted by $\mathcal{U}_P$ and B_P, are the set of ground terms and the set of ground atomic formulae of P, respectively. We denote the set of ground atomic formulae of P as $Fact(P)$. We call any fuzzy subset s of B_P the *fuzzy Herbrand model* of P.

Using the operator H_λ defined in Section 2.2, we define the following family of operators $(T_{P,\lambda})_{\lambda \in [0,1]}$ on B_P:

$$\begin{aligned} T_{P,\lambda}(X) \;=\; & H_\lambda(X) \cup Fact(P) \cup \{\beta \mid \\ & \beta \leftarrow B_1, \ldots, B_n \in |P|, B_1, \ldots, B_n \in X\} \end{aligned}$$

Where $|P|$ is the set of ground instantiations of the clauses in P. The operator $T^n_{P,\lambda}$ is defined as usual: $T^0_{P,\lambda} = i$, and for any integer i, $T^n_{P,\lambda} = T_{P,\lambda}(T^{n-1}_{P,\lambda})$.

For example:

$$T^1_{P,\lambda}(\emptyset) = Fact(P)$$
$$T^2_{P,\lambda}(\emptyset) = H_\lambda(Fact(P)) \cup \{\beta | \beta \leftarrow B_1, \ldots, B_n \in |P|, B_1, \ldots, B_n \in Fact(P)\}$$
$$T^3_{P,\lambda}(\emptyset) = H_\lambda\left(T^2_{P,\lambda}(\emptyset)\right) \cup \left\{\beta \mid \beta \leftarrow B_1, \ldots, B_n \in |P|, B_1, \ldots, B_n \in T^2_{P,\lambda}(\emptyset)\right\}\Big)$$

Obviously, for any $\lambda \in [0, 1]$, $T_{P,\lambda}$ is a continuous operator.

DEFINITION 14. *Let P be a definite program. Let $\lambda \in [0, 1]$. The family of operators $J_{P,\lambda} : \mathcal{P}(B_P) \to \mathcal{P}(B_P)$ is defined by setting, for any $X \in \mathcal{P}(B_P)$:*

$$J_{P,\lambda}(X) = \bigcup_{n \in N} T^n_{P,\lambda}(X)$$

On account of Theorem 1, the following fuzzy operator on $\mathcal{F}(B_P)$ defined by setting, for any $s \in \mathcal{F}(B_P)$ and for any $y \in s$, is a stratified fuzzy closure operator:

$$J_P(s)(y) = \vee\{\lambda \in [0, 1] \mid y \in J_{P,\lambda}(C(s, \lambda))\}$$

DEFINITION 15. *Let P be a definite program. We define the extended least Herbrand model of P as, the fuzzy subset $J_P(\emptyset)$ and denote it M_P.*

We now prove that for a definite program, P, the extended least Herbrand model is equal to the fuzzy subset $\mathcal{D}_P$ of the e-refutations.

THEOREM 5. *Let P be a definite program. Then*:

$$M_P = \mathcal{D}_P$$

Proof. We prove that, for any $\lambda \in [0, 1]$,

$$\mathcal{C}(\mathcal{D}_P, \lambda) = J_{P,\lambda}(\emptyset)$$

Namely, $\mathcal{C}(\mathcal{D}_P, \lambda) \supseteq J_{P,\lambda}(\emptyset)$ and $\mathcal{C}(\mathcal{D}_P, \lambda) \subseteq J_{P,\lambda}(\emptyset)$.

$\supseteq$ Suppose that a formula α is in $J_{P,\lambda}(\emptyset)$. Then a minimum integer n_α exists such that $\alpha \in T^{n_\alpha}_{P,\lambda}(\emptyset)$. We proceed by induction on n_α:

- Suppose that $n_\alpha = 1$. Then, if $\alpha \in T_{P,\lambda}(\emptyset) = Fact(P)$, there exists a classic refutation of α. Therefore, $\alpha \in \mathcal{C}(\mathcal{D}_P, \lambda)$ for any $\lambda \in [0, 1]$.
- Suppose the thesis to be true for $n_\alpha \leq m - 1$. We prove that, if $\alpha \in T^m_{P,\lambda}(\emptyset)$, then $\alpha \in \mathcal{C}(\mathcal{D}_P, \lambda)$. Since $T^m_{P,\lambda}(\emptyset) = H_\lambda(T^{m-1}_{P,\lambda}(\emptyset)) \cup \{\beta \mid \beta \leftarrow B_{1,\ldots,} B_n \in |P|, B_{1,\ldots,} B_n \in T^{m-1}_{P,\lambda}(\emptyset)\}$, then two cases are possible:
  (1) $\alpha \in H_\lambda(T^{m-1}_{P,\lambda}(\emptyset))$, or
  (2) $\alpha \in \{\beta \leftarrow B_{1,\ldots,} B_n \in P, B_{1,\ldots,} B_n \in T^{m-1}_{P,\lambda}(\emptyset)\}$.

In case (1), there exists a formula $\beta \in B_P$ such that $\beta \in T^{m-1}_{P,\lambda}(\emptyset)$ and $\mathcal{R}(\alpha, \beta) \geq \lambda$. By induction hypothesis, $\beta \in \mathcal{C}(\mathcal{D}_P, \lambda)$. Therefore, $\mathcal{D}_P(\beta) \geq \lambda$. By the extensionality of $\mathcal{D}_P$, $\mathcal{D}_P(\alpha) \geq R(\alpha, \beta) \wedge \mathcal{D}_P(\beta) \geq \lambda$. Therefore, $\alpha \in \mathcal{C}(\mathcal{D}_P, \lambda)$.

In case (2), there exists a ground instantiation $\alpha \leftarrow B_{1,\ldots,} B_n$ of a clause in P such that $B_{1,\ldots,} B_n \in T_{P,\lambda}^{m-1}(\emptyset)$. By induction hypothesis, $B_{1,\ldots,} B_n \in \mathcal{C}(\mathcal{D}_P, \lambda)$. Consequently, for any $i = 1, \ldots, n$, there exists an e-refutation r_i of the e-formula B_i such that $h(r_i) \geq \lambda$. Therefore, the sequence $(r_i)_{i=1,\ldots,n} \cup \alpha \leftarrow B_{1,\ldots,} B_n$ is an e-refutation, γ, of α such that $h(\gamma) = \bigwedge_{i=1,\ldots,n} h(r_i) \wedge 1 \geq \lambda$. The thesis follows immediately.

$\subseteq$ For any $\lambda \in [0, 1]$, we set two equivalence relations, $\mathcal{R}_{\lambda,\mathcal{C}}$ and $\mathcal{R}_{\lambda,P}$, on $\mathcal{C}$ and P, respectively, as follows: for any $c, c' \in \mathcal{C}$ and $p, p' \in P$:

$$c \,\mathcal{R}_{\lambda,\mathcal{C}}\, c' \quad \text{if } \mathcal{R}_{\mathcal{C}}(c, c') \geq \lambda$$
$$p \,\mathcal{R}_{\lambda,P}\, p' \quad \text{if } \mathcal{R}_{P}(p, p') \geq \lambda$$

Let $\mathcal{C}_\lambda = \mathcal{C}_{/\lambda}$ and $P_\lambda = P_{/\lambda}$, and let $\mathcal{L}_\lambda$ be the related first-order language; then, for any formula α in $\mathcal{L}$, we denote α_λ in the following way:

$$\alpha_\lambda = \bar{p}(\bar{t}_1, \ldots, \bar{t}_n) \quad \text{if } \alpha = p(t_1, \ldots, t_n) \in \mathcal{L},\ \bar{p} \in P_\lambda, \bar{t}_i \in \mathcal{C}_\lambda \text{ for any } i$$

Therefore, we can establish a natural correspondence between $\mathcal{L}$ and $\mathcal{L}_\lambda$. As a consequence, to any definite program P in $\mathcal{L}$, there corresponds a program $\mathcal{P}_\lambda$ in $\mathcal{L}_\lambda$ such that, denoting by $D_{\mathcal{P}_\lambda}$ and $J_{\mathcal{P}_\lambda}(\emptyset)$ the success set and the least-fixed point of $\mathcal{P}_\lambda$, respectively, we have:

$$\alpha \in \mathcal{C}(D_P, \lambda) \Rightarrow \alpha_\lambda \in D_{\mathcal{P}_\lambda}$$
$$\alpha_\lambda \in J_{\mathcal{P}_\lambda}(\emptyset) \Rightarrow \alpha \in J_{P,\lambda}(\emptyset)$$

Hence, by a result proven in Ref. 12, we have:

$$\alpha \in C(D_{\mathcal{P}}, \lambda) \Rightarrow \alpha_\lambda \in D_{\mathcal{P}_\lambda} \Leftrightarrow \alpha_\lambda \in J_{\mathcal{P}_\lambda}(\emptyset) \Rightarrow \alpha \in J_{P,\lambda}(\emptyset) \qquad \blacksquare$$

Through Theorem 5, we give the completeness result, which proves the coincidence of the operational and fixed-point semantics of our extended logic programming language.

6. INTRODUCING SIMILARITY IN A DATABASE

Deductive databases can be viewed as information retrieval systems in which knowledge is stored both explicitly, through facts, and implicitly, through a set of rules. Data are extracted through a *query evaluation* method. Our main goal in the study of similarity-based unification is the quest to develop system models capable of giving a flexible answer to a query. This is the premise of fuzzy logic programming.

As an example, consider the following program P that would reference a database of books that would be classified on two features. The books would be classified by genre and cost. Sample classifications might be:

adventurous(isle_of_treasure)
adventurous(murder_on_orient_express)
war(operation_overlord)
not_expensive(isle_of_treasure)

best_seller(operation_overlord)
adventurous(james_bond)
cost(isle_of_treasure,6)
cost(the_fly,9)
good(X) :- interesting(X),cheap(X)
cheap(X) :- cost(X,L), L <= 10

The goal "good"(X) has no classic solution in an ordinary program, because no element in the database unifies with the conclusion "interesting"(X). Nevertheless, it seems reasonable to consider the constants "interesting" and "adventurous" as "similar" to a certain degree. Similarly, one would generally consider "not_expensive" and "cheap" as similar.

To address this shortcoming, we could provide P with the following evaluations based on the degree to which they relate:

R(adventurous,interesting)=0.9
R(adventurous,horror)=0.7
R(war,interesting)=0.7
R(good,best_seller}=0.8
R(cheap,not_expensive)=1

The query "good(X)" can be evaluated with a similarity-based derivation, giving as a result the answer X→ James_Bond, provided that "adventurous" is similar to "interesting."

Under the same constraints, the following answers are acceptable:

X → isle_of_treasure
X → murder_on_orient_express

The answer X → isle_of_treasure could be acceptable provided that "cheap" is considered to be synonymous with "not_expensive" as well.

The answer X → operation_overlord could be obtained in two ways: one can consider "interesting" and "war" as similar, or one can consider "good" and "best_seller" as similar. In this case, the degree of the answer is the maximum of the degree of similarity among the two pairs of terms, that is, 0.9.

We must observe that the predicate "good" does not stand for an absolute or objective judgment on the quality of a book. Ultimately, the subjective nature of the program cannot be avoided, as it is up to the holder of the library, or to the user of the database, to formulate the similarity. Thus, the predicate "good" might mean "good_for_me", or "good_for_the user," accordingly, and depending on the manner in which evaluations are assigned to the parameters that might be used to identify the books or other identified elements of the database.

The same result can be achieved computing the value assumed by the fuzzy subset $J_P(\emptyset)$ on the corresponding atomic formula. Indeed, for any $\lambda \leq 0.9$, we have that: `good(isle_of_treasure)` $\in H_\lambda(Fact(P))$.

Since `cost(isle_of_treasure,6)` $\in H_\lambda(Fact(P))$, it follows that:

$$\texttt{good(isle\_of\_treasure)} \in J_{P,\lambda}(\emptyset) \text{ for any } \lambda \leq 0.9$$

As a consequence:

$$J_P(\emptyset)(\texttt{good(isle\_of\_treasure)}) = D_P(\texttt{good(isle\_of\_treasure)}) = 0.9$$

Hence, as asserted in Theorem 5, we have that the degree of refutation of the formula `good(isle_of_treasure)` is equal to the degree of membership of `good(isle_of_treasure)` to the fuzzy least Herbrand model of P.

The example above could be more structured by introducing a similarity not only on predicates, but also on constants. For example, one can define the predicate `costs(book,N)` and set a similarity over the integer numbers.

6.1. The Choice of a Suitable t-Norm

It must be observed that, as mentioned earlier in Section 2, the t-norm of the minimum is not always the best choice for modeling users' preferences.

For example, let us consider the case in which a user thinks that "thriller" and "horror" are two completely different literary genres. In this case, the user will reasonably assume that $\mathcal{R}(\text{thriller, horror}) = 0$. According to the minimum, this will imply that $\mathcal{R}(\text{thriller, adventurous}) = \mathcal{R}(\text{horror, adventurous}) = 0$, which does not seem plausible.

The reason for this problem is due to the use of the t-norm of the minimum. If we adopt the Lukasiewicz t-norm, then we could set $\mathcal{R}(\text{thriller, horror}) = 0$ and nevertheless define $\mathcal{R}(\text{thriller, adventurous}) = \mathcal{R}(\text{horror, adventurous}) = 0.3$. This is more reasonable, because a user that chooses the Lukasiewicz t-norm might be seen as someone that doesn't see things "similar" beyond a certain threshold, but similar only in a certain context.

7. CONCLUSIONS

In this article, we have described a new kind of fuzzy resolution based on similarity. We introduced a similarity in a first-order language, and then, starting from the extension of unification theory, proposed an extended resolution rule. This constitutes the kernel of a fuzzy logic programming language, called Likelog, and we provided the operational and declarative semantics of this language.

We are planning to use other t-norms for the similarity, such as the natural product in [0, 1] and the Lukasiewicz sum. We have not considered the class of fuzzy Herbrand models as a fuzzy semantics. To go in this direction, it should be necessary to extend a fuzzy Herbrand model to the whole set of formulae in a program P. We showed an example in which we exploit similarity for flexible data retrieval in deductive databases; our system can be used in several other fields as well, such as in information filtering and evaluation, distributed data mining, and digital libraries.

Acknowledgments

We are grateful to the anonymous referees for their valuable remarks. We wish to thank Miss Sierra Shain for her help in improving the English of this article.

References

1. Apt KR. Logic programming. In: van Leeuvwen J, editor. Handbook of theoretical C.S., volume B, chapter 10. Elsevier Science; 1990. pp 495–574.
2. Arcelli F, Formato F, Gerla G. Fuzzy unification as a foundation of fuzzy logic programming. In: Martin TP, Arcelli F, editors. Logic programming and soft computing—Uncertainty theory in artificial intelligence series. England: Research Studies Press, Wiley; 1998.
3. Arcelli F, Formato F. LIKELOG: A logic programming language for flexible data retrieval. In: Proc ACM—SAC'99. San Antonio, TX; March 1999. pp 260–267.
4. Aronson AR, Kacobs BE, Minker J. A note on fuzzy deduction. JACM 1980;27(4):599–603.
5. Baldwin JF, Martin TP, Pilsworth BW. FRIL—Fuzzy and evidential reasoning in AI. Research Studies Press; 1995.
6. Baldwin JF. Evidential support logic programming. Fuzzy Sets Syst 1987;(24):1–26.
7. Bosc P, Pivert O. Discriminated answers and databases: Fuzzy sets as a unifying expression means. In: Proc FUZZ-IEEE92. San Diego; 1992. pp 745–752.
8. Dubois D, Lang J, Prade H. Towards possibilistic logic programming. In: Proc ICLP91. Paris; 1991.
9. Dubois D, Lang J, Prade H. Automated reasoning using possibilistic logic: Semantics, belief revision and variable certainty weights. IEEE Trans Knowledge Data Eng 1994;5(1):64–71.
10. Formato F. Similarity in logic programming. PhD thesis, University of Naples; 1999.
11. Gerla G. An extension principle for fuzzy logic. Mathematical Logic Quarterly 1984;(40):357–380.
12. Gerla G, Sessa MI. Similarity and like-prolog. In: Proc Gulp 99. L'Aquila; June 1999.
13. Lee RCT. Fuzzy logic and the resolution principle. JACM 1972;(19):109–119.
14. Martin TP, Arcelli F. Logic programming and soft computing—Uncertainty theory in artificial intelligence series. England: Research Studies Press, Wiley; 1998.
15. Martelli A, Montanari U. An efficient unification algorithm. ACM Trans Programming Languages Syst 1982;4(2):258–282.
16. Mukaidono M. Fuzzy inference of resolution style. In: Yager RR, editor. Fuzzy sets possibility theory. New York: Pergamon Press; 1982. pp 224–231.
17. Robinson JA. A machine oriented logic based on the resolution priciple. JACM 1965;12(1):23–41.
18. Shen Z, Ding L, Mukaidono M. A theoretical framework of fuzzy prolog machine. In: Gupta MM, Yamakawa T, editors. Fuzzy computer. Amsterdam: North-Holland; 1988.
19. Virtanen H. A study in fuzzy logic programming. In: Proc 12th European Meeting on Cybernetics and Systems '94. Austria; April 1994. pp 249–256.
20. Weigert TJ, Tsai J, Liu X. Fuzzy operator logic and fuzzy resolution. J Automat Reas 1993;10:59–78.
21. Yasui HY, Hamada Y, Mukaidono M. Fuzzy prolog based on Lukasiewicz implication and bounded product. In: Martin TP, Arcelli F, editors. Logic programming and soft computing—Uncertainty theory in artificial intelligence series. England: Research Studies Press, Wiley; 1998.