Towards the Theory of Relaxed Unification

Tony Abou-Assaleh, Nick Cercone, and Vlado Keselj

Faculty of Computer Science, Dalhousie University,
Halifax, Ontario, B3H 1W5, Canada,
{taa,nick,vlado}@cs.dal.ca

Abstract. Classical unification is strict in the sense that it requires a perfect agreement between the terms being unified. In practice, data is seldom error-free and can contain inconsistent information. Classical unification fails when the data is imperfect. Fuzzy unification partially addresses uncertainty but requires the creation of fuzzy databases. We propose the *Theory of Relaxed Unification* as a new theory that relaxes the constraints of classical unification without requiring special pre-processing of data. The goal of relaxed unification is to tolerate possible errors and inconsistencies in the data and facilitate reasoning under uncertainty. The Theory of Relaxed Unification is more general and has higher expressive power than the classical Theory of Unification. We present the fundamental concepts of relaxed unification, a relaxed unification algorithm, and a number of examples.

1 Introduction

The classical unification function takes two terms as input and produces a boolean value indicating whether the unification can be performed successfully. In case of a result of true, the function also returns a substitution that unifies these two terms. The unification fails if the same feature is assigned different values in the objects being unified. This process places rigid constraints on the data requiring it to be correct and consistent. Since real-world data is seldom perfect, classical unification fails when it encounters the slightest error. Erroneous data often contains enough information that one can exploit to overcome the errors. In other cases, it is possible to draw approximate or uncertain conclusions.

Fuzzy unification is useful in situations when the exact values of some attributes are not known. To benefit from fuzzy unification, the uncertain values in the data must be described using approximate qualitative attribute values. Such data is normally stored in fuzzy databases. However, fuzzy unification still requires consistency between the approximate values. A single erroneous instance in the data set causes the whole unification process to fail.

Probabilistic logic is encapsulates the probability theory with first-order logic. It provides a mechanism for specifying different degrees of belief and evidence to propositions. Although this area has been investigated for several decades, it is still under development. Inconsistencies in the data set are problematic.

Relaxed unification provides a method for extracting information from imperfect data. To achieve this functionality, we relax the strict true/false result

of classical unification and replace it by a real number in the range $(0, 1]$ that indicates the correctness of the unification. A correctness value of 1 would represent a success under the classical unification; any other value would represent a failure. Relaxed unification does not includes a notion of failure; the unification always succeeds and returns a substitution.

2   Related Work

The Theory of Unification is a well formalized and understood. Robinson [12] was the first to introduce unification in 1965. He set forth the basic definitions of the theory, presented a straightforward recursive algorithm for unifying two terms, and proved some theorems and lemmas that are fundamental to the Theory of Unification. Knight [8] provided an extensive survey of representations, algorithms, and applications of unification. More recently, Keselj [7] introduced an efficient general-purpose graph unification algorithm and discussed the low-level details of its implementation. Although classical unification has witnessed a great success, its assumption that the data is absolutely true isolated it from any real-world problems that involve uncertainty.

Lee [9] was one of the leading figures who introduced uncertainty into the realm of logic programming. His notion of Fuzzy Resolution, which was based on the Fuzzy Set Theory, has opened the door for Fuzzy Logic and the Fuzzy Unification Theory. Significant work in advancing Fuzzy Logic was done by Mukaidono [10] and Baldwin [5].

Several attempts have been made to facilitate reasoning under uncertainty by defining probabilities over first-order languages. These attempts lacked a unified representation model and were limited in their expressive power. Bacchus [4] separated probabilities into statistical and propositional probabilities, which allowed him to represent and perform probabilistic inference in a natural and unified formalism.

Multi-valued attributes have had a limited success in unification [11, 6], however, their use was specialized and was not developed into a complete general theory. Relaxed unification [1, 2] was the first attempt to formalize the concept of unifying sets of values. Further work has led to proposing the Theory of Relaxed Unification [3], which is the first coherent and complete formalization of the theory along with an implementation of a relaxed unification system.

3   Classical Unification

We present the fundamental definitions in the Theory of Unification. A discussion of representation issues and unification algorithms can be found in [8].

3.1   Definitions

Definition 1 (Classical Function). *A function is an element from a countably infinite set of functions F. Each function has zero or more arguments.*

Each argument is a classical term. A function symbol is an element from the set $\Sigma_F = \{f, g, h, \ldots\}$.

Definition 2 (Classical Constant). *A constant is an element from a countably infinite set of nullary functions (functions that do not have arguments) A, i.e, $A \subset F$. A constant symbol is an element from the set $\Sigma_A = \{a, b, c, \ldots\}$.*

Definition 3 (Classical Variable). *A variable is an element from the infinite set of variables V. A variable can be substituted by any classical term. A variable symbol is an element from the set $\Sigma_V = \{x, y, z, \ldots\}$.*

Definition 4 (Classical Term). *We construct a term algebra $T(F, V)$ from the set of functions F and the set of variables V. Thus, a term can be a constant, a variable, or a function. A term symbol is an element from the set $\Sigma_T = \{t, u, v, \ldots\}$. E.g., $t = a$, $u = x$, and $v = f(x, a, h(b))$.*

Definition 5 (Classical Substitution). *A substitution is a mapping from variables to terms. Usually substitutions contain a finite number of mapping rules. A substitution symbol is one of $\{\sigma, \tau, \theta, \ldots\}$. E.g., $\sigma = \{x \mapsto a, y \mapsto f(a)\}$. The application of a substitution σ to a term t, denoted as $t\sigma$, is defined as,*

$$
t\sigma = \begin{cases} u & \text{if } t = x \text{ and } x \mapsto u \in \sigma \\ f(t_1\sigma, \ldots, t_n\sigma) & \text{if } t = f(t_1, \ldots, t_n) \\ t & \text{otherwise} \end{cases} \quad .
$$

Basically, we descend through function arguments looking for variables that have corresponding mappings rule in the substitution and replace them with the substitute term. E.g., if $\sigma = \{x \mapsto h(a), y \mapsto b\}$ and $t = f(x, g(y), z)$ then $t\sigma = f(h(a), g(b), z)$.

Substitutions can be composed by applying one substitution to another. The application of substitution τ to σ, denoted $\sigma\tau$, is achieved by applying τ to the term of each mapping rule in σ and adding the mapping rules in τ to σ if they are not already in σ. Substitution combination is associative, i.e., $(\sigma\tau)\theta = \sigma(\tau\theta)$ but generally is not commutative, i.e., $\sigma\tau \neq \tau\sigma$.

Definition 6 (Classical Unification). *Two terms, t and u, are unifiable if and only if there exists a unifying substitution σ such that $t\sigma = u\sigma$. The term resulting from the unification is denoted $t \sqcup u$. In other words:*

$$
t \sqcup u \iff \exists \sigma : t\sigma = u\sigma
$$

The unifying substitution σ is called a unifier *for the terms t and u.*

Definition 7 (Most General Classical Unifier). *A unifier σ of terms t and u is the most general unifier if for any other unifier θ there exists a substitution τ such that $\sigma\tau = \theta$.*

4 Relaxed Unification

We begin with an intuitive introduction to relaxed unification and present it as a natural extension to classical unification. After a discussion of some of the representation conventions of relaxed terms we introduce the fundamental concepts of relaxed unification and give a formal definition to the proposed theory. Next, we present a high-level algorithm for relaxed unification. We conclude this section with a set of illustrative examples.

4.1 From Classical to Relaxed Unification

There is a fundamental limitation in classical unification that prevents it from being applied to numerous real life applications. The limitation follows directly from the definition of classical unification, which requires that corresponding functions in the unified terms be identical. As a result, classical unification is useless if the data is not perfectly consistent and absolutely true. Consider the following examples.

Example 1. Let t, u, and v be three terms where $t = f(x, h(g(b), a), b)$, $u = f(a, h(g(c), a), y)$, and $v = f(a, b, c)$. Although the term t is not unifiable with neither the term u nor v, the degree of the mismatch between t and u and between t and v is clearly different. In the case of $t \sqcup u$, the mismatch is only in a single element 3 functions deep, whereas in the case of $t \sqcup v$ the mismatch is in 2 arguments of the top-level function f. Classical unification returns the same result of failure in both cases without making this distinction. Let us examine what happens when we relax the unification rule and allow mismatched items to form a set. The relaxed unification of t and u, denoted $t \sqcup_R u$ results in $f(a, h(g(\{b, c\}), a), b)$ while the relaxed unification of t and v, denoted $t \sqcup_R v$ results in $f(a, \{h(g(b)), b\}, \{b, c\})$. We apply an evaluation function δ to both results. For simplicity, let the evaluation function be $\delta_{\bar{1}}$ — the number of non-singleton sets in the relaxed term. This evaluation function returns a numerical value that allows us to compare the degree of the mismatch between the unification terms. It is possible now to systematically conclude that $t \sqcup_R u$ is a better match, and potentially more accurate, than $t \sqcup_R v$ because $\delta_{\bar{1}}(t \sqcup_R u) = 1$ is smaller than $\delta_{\bar{1}}(t \sqcup_R v) = 2$. Note that $\delta_{\bar{1}}$ returns 0 if the terms are unifiable under classical unification, which implies that relaxed unification is more general than classical unification.

We encapsulate each function of a classical term by a singleton set that contains the function. E.g., the classical term $f(x, h(a), b)$ becomes the relaxed term $\{f(x, \{h(\{a\})\}, \{b\})\}$. Since the relaxed term consists of sets of functions, each set may contain any number of elements and not only one, e.g, $\{f(\{a, h(\{b\}), c\})\}$. To make relaxed terms as general as possible, subterms can be shared. Structure sharing includes sharing of functions, variables, and sets. The next section details the representation aspects of relaxed terms, both symbolic and graphical.
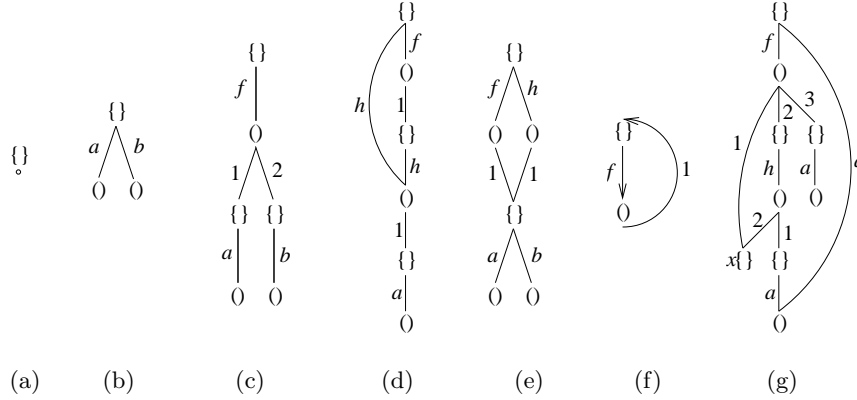
Fig. 1. Relaxed term representation: (a) an empty set; (b) set elements; (c) function arguments; (d) shared function; (e) shared set; (f) recursive structure; and (g) combination with variables

4.2 Representation

Symbolically, shared structures are represented by suffixing a boxed index to the elements being shared. Graphically, A relaxed term is represented as a directed rooted graph. The nodes in the graph represent sets and functions. Edges directed from set nodes to function nodes are labelled with the function symbols of the elements of the set. Edges directed from function nodes to set nodes are labelled with the positional index of the arguments, starting from 1. Variables are represented in the graph by empty sets with the variable symbol attached to the label of the set node. Sets cannot contain sets or variables as elements. The following example presents the symbolic representations of a number of relaxed terms. The graphical representation of these terms is shown in Figure 1.

Example 2.

(a) Empty Set. $t = \{\}$.
(b) Set Elements. $t = \{a, b\}$.
(c) Function Arguments. $t = \{f(\{a\}, \{b\})\}$.
(d) Shared Functions. $t = \{h(\{a\})\,\boxed{1}\,, f(\{h\,\boxed{1}\,\})\}$.
(e) Shared Sets. $t = \{f(\{a, b\}\,\boxed{1}\,), h(\{\}\,\boxed{1}\,)\}$.
(f) Recursive Structure. $t = \{f(\{\}\,\boxed{1}\,)\}\,\boxed{1}\,$.
(g) Combination With Variables. $t = \{a\,\boxed{1}\,, f(x, \{h(\{a\,\boxed{1}\,\}, x), a\})\}$.

4.3 Definitions

Definition 8 (Relaxed Function). *A function is an element from a countably infinite set of functions F. Each function has zero or more arguments. Each*

argument is a relaxed term. A function symbol is an element from the set $\Sigma_F = \{f, g, h, \ldots\}$.

Definition 9 (Relaxed Constant). *A constant is an element from a countably infinite set of nullary functions (functions that do not have arguments) A, i.e, $A \subset F$. A constant symbol is an element from the set $\Sigma_A = \{a, b, c, \ldots\}$.*

Definition 10 (Relaxed Variable). *A variable is an element from the infinite set of variables V. A variable can be substituted by any relaxed term. A variable symbol is an element from the set $\Sigma_V = \{x, y, z, \ldots\}$.*

Definition 11 (Relaxed Term). *We construct a term algebra $T(S_F, V)$ where S_F is a set of all of the subsets of the set of functions F and V is the set of variables. A term can be an empty set $\emptyset = \{\}$, a set of constants and functions, or a variable. A term symbol is an element from the set $\Sigma_T = \{t, u, v, \ldots\}$. Structure sharing is allowed and is represented symbolically by suffixing a boxed index to the shared elements. The graphical representation is a directed rooted graph where each set and function is represented by a node. Since each node is the root of a subterm, we use the terms* node *and* subterm *interchangeably.*

Definition 12 (Perfect Term). *A perfect term is a relaxed term in which all sets are either empty or singleton. Each classical term has a corresponding perfect term that is formed by encapsulating each function in a set.*

Definition 13 (Path). *A path to a node u in a term t, denoted as $\pi(u)$, is a sequence of edges that connects the root of the graph of t to u. The set of all paths from the root to u is denoted as $\Pi(u)$.*

Definition 14 (Relaxed Substitution). *A substitution is a mapping from variables to sets of functions and from sets of functions to sets of functions. A substitution symbol is one of $\{\sigma, \tau, \theta, \ldots\}$. The application of a substitution σ to a term t, denoted as $t\sigma$, is defined as*

$$
t\sigma = \begin{cases} u & \text{if } t = x \text{ and } x \mapsto u \in \sigma \\ u & \text{if } \Pi(v) \cap \Pi(t) \neq \emptyset \text{ and } v \mapsto u \in \sigma \\ f(t_1\sigma, \ldots, t_n\sigma) & \text{if } t = f(t_1, \ldots, t_n) \\ \{t_1\sigma, \ldots, t_n\sigma\} & \text{if } t = \{t_1, \ldots, t_n\} \\ t & \text{otherwise} \end{cases} .
$$

Basically, the mapping rule is applied if it matches the node. Otherwise we recursively descend through function arguments and set elements.

Substitutions can be combined by applying one substitution to another. The application of a substitution τ to σ, denoted $\sigma\tau$, is achieved by applying τ to the term of each mapping rule in σ and merging the mapping rules in τ with the mapping rules in σ. The merging is done by adding a mapping rule from τ to σ if σ does not contain a mapping rule with the same right-hand-side. If σ does contain such a mapping rule then the left-hand-sides of the two mapping rules are relax unified. Substitution combination is associative, i.e., $(\sigma\tau)\theta = \sigma(\tau\theta)$ but generally is not commutative, i.e., $\sigma\tau \neq \tau\sigma$.

Definition 15 (Relaxed Unification). *Two terms, t and u, are always unifiable with a unifying substitution σ such that $t\sigma = u\sigma$. The term resulting from the unification is denoted $t \sqcup_R u$. I.e.,*

$$t \sqcup_R u \iff \exists \sigma : t\sigma = u\sigma \ .$$

The unifying substitution σ is called a relaxed unifier *for the terms t and u and must meet the following restriction. For each mapping $v \mapsto w \in \sigma$, if v is a set node then $w = v_t \cup v_u$ where v_t is the corresponding set of v in t and v_u is the corresponding set of v in u. This restriction is necessary to prevent substitutions such as $\sigma = \{[rootset] \mapsto \emptyset\}$ from unifying terms by changing their structure to a predefined term independent of the terms being unified and consequently losing all the information associated with them.*

Definition 16 (Most General Relaxed Unifier). *A unifier σ of terms t and u is the most general unifier if for any other unifier θ there exists a substitution τ such that $\sigma\tau = \theta$.*

Definition 17 (Evaluation Function). *An evaluation function maps relaxed terms to a real number. An evaluation function symbol is an element from the set $\Delta = \{\alpha, \beta, \delta, \ldots\}$.*

Definition 18 (Correctness Function). *A correctness function δ is an evaluation function that meets the following criteria:*

1. *the range of δ is $(0, 1]$;*
2. *for any two relaxed terms t and u, $\delta(t) > \delta(u) \iff t$ is more accurate than u according to some user-defined measure of correctness; and*
3. *for any relaxed term t, $\delta(t) = 1 \iff t$ is a perfect term.*

4.4 Algorithm

A high-level algorithm for relaxed unification is provided below as the function RelaxUnify (Algorithm 1). A low-level algorithm that addresses implementation issues with respect to handling shared and recursive structures can be found in [3]. It takes the roots of the terms to be unified as arguments u and v and returns the root of the unified graph t and a unifier σ. The RelaxUnify function starts by applying the substitution to u and v. There are two possible actions. If u and v are set nodes then $t = u \sqcup_R v$ is a set node where the set is the union of the u and v sets. The union operation generates a new set if u and v do not contain the same elements, in which case a mapping is added to σ to reflect this change. During the merging of u and v, equal symbols are recursively relax unified. The second possibility is that u and v are equal function symbols (a call generated recursively from the previous action). In this case t is a function with the same function symbol as u and v. Arguments of t are obtained by recursively relax unifying the corresponding arguments of u and v. The recursion terminates when a constant is reached or when u and v are sets and $u \cap v = \emptyset$.

Algorithm 1 *Relaxed Unification by Recursive Descent Algorithm*

global σ : Substitution := $\emptyset$
function RelaxUnify(u : Node, v : Node) $\Rightarrow$ (t : Node, σ : Substitution)
begin
  $u := u\sigma$
  $v := v\sigma$
  if $u = \{u_1, u_2, \ldots, u_n\}$ **and** $v = \{v_1, v_2, \ldots, v_m\}$ for $n, m \geq 0$ **then**
    $t :=$ merge u and v unifying edges with identical labels
    if $t \neq u$ **then** $\sigma := \sigma\{u \mapsto t\}$
    if $t \neq v$ **then** $\sigma := \sigma\{v \mapsto t\}$
  else $u = f(u_1, u_2, \ldots, u_n)$, $v = f(v_1, v_2, \ldots, v_n)$ for $n \geq 0$
    $t := f(t_1, t_2, \ldots, t_n)$ where $t_i :=$ RelaxUnify(u_i, v_i)
  end if
  return (t, σ)
end

    An external user-defined correctness function is used to determine the correctness value of the resulting term. This function can be as simple as the number of empty and singleton sets divided by the total number of sets in the term. A smoother and slightly more sophisticated correctness function is to associate with each nonempty set a correctness value equal to the inverse of the cardinality of the set and with each empty set a correctness value of 1.0. The correctness value of the whole term is the average of the correctness values of each of the sets in the term. A more complex correctness function would take into account the depth of the nodes being evaluated. The user must decide whether shared structures are counted once or every time they are referenced and how to evaluate recursive structures depending on the semantics of the relaxed term in a particular application.

4.5 Examples

Boxed indices are used to identify nodes in general, which allows referring to them in unifiers.

Example 3. Functions with different arguments.

$$t = \{f(\{b\}^{\boxed{1}})\}$$
$$u = \{f(\{a\}^{\boxed{1}})\}$$
$$t \sqcup_R u = \{f(\{a, b\}^{\boxed{1}})\}, \sigma = \{\boxed{1} \mapsto \{a, b\}\}$$

Example 4. Variables.

$$t = \{f(x^{\boxed{1}})\}$$
$$u = \{f(\{a\}^{\boxed{1}})\}$$
$$t \sqcup_R u = \{f(\{a\}^{\boxed{1}})\}, \sigma = \{x \mapsto \boxed{1}, \boxed{1} \mapsto \{a\}\}$$

The unifier always maps the variables to their positions in the term and then maps the those positions to an appropriate set.

Example 5. Shared structure at the set level.

$$t = \{f(\{b\}\,\boxed{1}), h(\{\}\,\boxed{1})\}$$
$$u = \{f(\{a\}\,\boxed{1}), h(\{c\}\,\boxed{2})\}$$
$$t \sqcup_R u = \{f(\{a,b,c\}\,\boxed{1}), h(\{\}\,\boxed{1})\}, \sigma = \{\boxed{1} \mapsto \{a,b,c\}, \boxed{2} \mapsto \boxed{1}\}$$

The term t contains a shared structure while the term u does not (indicated by different node indices in u). The structure sharing information is propagated to the unification term $t \sqcup_R u$ and $\{b\}$ is unified with both $\{a\}$ and $\{c\}$ resulting in $\{a,b,c\}$. The unifier σ enforces the structure sharing when applied to u by encoding it in the mapping $\boxed{2} \mapsto \boxed{1}$.

Example 6. Infinite unification.

$$t = x$$
$$u = \{f(x)\}$$
$$t \sqcup_R u = \{f(\{\}\,\boxed{1})\}\,\boxed{1}, \sigma = \{x \mapsto \boxed{1}\}$$

The terms t and u fail to classically unify but are naturally relax unifiable forming a recursive structure.

Example 7. Recursive structure at the function level.

$$t = \{f(\{f\,\boxed{1}\}\,\boxed{2})\,\boxed{1}\}$$
$$u = \{f(\{a\}\,\boxed{2})\}$$
$$t \sqcup_R u = \{f(\{a, f\,\boxed{1}\}\,\boxed{2})\,\boxed{1}\}, \sigma = \{\boxed{2} \mapsto \{a, f\,\boxed{1}\}\}$$

5 Conclusion

Throughout this work, we propose a new unification theory that we call the *Theory of Relaxed Unification*. The purpose of this novel theory is to enable reasoning under uncertainty. Relaxed unification emerged from the limitation of classical unification that requires a perfect match between the unified terms. Relaxed unification succeeds where classical unification fails. In fact, relaxed unification always succeeds, which allows for reasoning in contradicting, erroneous, and uncertain data sets, as well as in consistent data sets.

One can use a *correctness function* in relaxed unification to evaluate relaxed terms and determine how accurate they are according to an accuracy semantic defined by the user. Terms can be unifiable at different degrees depending on the degree of the mismatch between them. Classical unification can be emulated in relaxed unification by interpreting a correctness value of 1.0 as true (success) and all the other values as false (failure). Thus, relaxed unification is more general than classical unification. We set forth the fundamental concepts of the Theory of

Relaxed Unification through concise formal definitions and present a unification algorithm.

This work is the first step towards developing the Theory of Relaxed Unification into a complete coherent theory. To uncover the full potential of relaxed unification, probabilities can be assigned to nodes. Stochastic relaxed unification is our main goal of future work.

Some of the specific theoretical aspects that we will address are: proving the correctness and effectiveness of the relaxed unification algorithm; deriving the runtime and space complexities of the relaxed unification algorithm; developing probabilistic models for assigning weights and probabilities to attributes and values and providing meaningful interpretations for these models; and deriving a scheme for computing the stochastic correctness of the unified objects.

We plan to incorporate stochastic relaxed unification into logic programming. We are currently building a stochastic logic programming environment with a probabilistic inference engine, pruning based on probability thresholds, and backtracking. This environment would allow writing logic programs that can reason under uncertainty and inconsistency of data.

References

1. Abou-Asslaeh, T. and Cercone, N.: Relaxed Unification – Proposal. In: R. Cohen and B. Spenser (Eds.), Lect. Notes Artif. Int.: 15th Conference of the Canadian Society for Computational Studies of Intelligence, AI 2002. Springer (2002)
2. Abou-Assaleh, T. and Cercone, N.: Relaxed Unification – Proposal. Appl. Math. Lett. Elsevier Science (to appear)
3. Abou-Assaleh, T.: Theory of Relaxed Unification – Proposal. Master Thesis. School of Computer Science, Univeristy of Waterloo, Waterloo, Ontario, Canada (2002)
4. Bacchus, F.: Representing and Reasoning with Probabilistic Knowledge. The MIT Press (1990)
5. Baldwin, J.F.: Evidential Support Logic Programming. Fuzzy Set. Syst. **24** (1987) 1–26
6. Carpenter, B.: The Logic of Typed Feature Structures. Cambridge Tracts in Theoretical Computer Science **32**, Cambridge University Press (1992)
7. Keselj, V. and Cercone, N.: A Graph Unification Machine for NL Parsing. Comput. Math. Appl. (to appear)
8. Knight, K.: Unification: A Multidisciplinary Survey. ACM Comput. Surv. **21** (1) (1989) 93–124
9. Lee, R.C.T.: Fuzzy Logic and the Resolution Principle. J. ACM **19** (1) (1972) 109–119
10. Mukaidono, M.: Fuzzy Inference of Resolution Style. In: R.R. Yager (Ed.), Fuzzy Set and Possibility Theory, Pergamon Press (1982) 224–231
11. Pollard, C.J. and Moshier, M.A: Unifying partial descriptions of sets. In: P.P. Hanson (Ed.), Information, Language, and Cognition, University of British Columbia Press, Vancouver (1990) 285–322
12. Robinson, J.A.: A Machine-Oriented Logic Based on the Resolution Principle. J. ACM **12** (1) (1965) 23–41