ELSEVIER

Fuzzy Sets and Systems 98 (1998) 311–317

FUZZY sets and systems

Notes on implementing fuzzy sets in Prolog

Toshinori Munakata

Department of Computer and Information Science, Cleveland State University, Cleveland, OH 44115, USA

Received October 1995; revised November 1996

Abstract

Due to its unique characteristics, Prolog requires special techniques for implementing ordinary as well as fuzzy sets. This article presents a comparative overview of various strategies of representing and manipulating fuzzy sets in Prolog. There are two major approaches to implement fuzzy sets in Prolog. One is to incorporate fuzzy representations and operations on top of an existing Prolog. The second way is to develop a new extended Prolog language. This article discusses various methods based primarily on the first approach. The choice of a method depends on many factors, such as whether a database for fuzzy sets already exists, the type of applications, the type of fuzzy set operations performed, whether an implicit description of the elements is possible, the size of the database, and the required computation time. The following methods are discussed in this article: explicit description (finite), using lists; explicit description (finite), using a fact for every element; implicit description (finite, infinite); and other methods and extensions.

Keywords: Artificial intelligence; Logic programming; Prolog

1. Introduction

The field of AI has undergone rapid growth in diversification and practicality for the past decade. The repertoire of AI techniques, such as fuzzy systems and Prolog-based logic programming, has evolved and expanded for truly practical applications [14–17].

Prolog (PROgramming in LOGic) is one of the two major AI languages, the other being Lisp. It is a simple, yet powerful declarative symbolic language based on predicate logic, having many applications. Prolog is particularly suitable when a problem is expressed in form of logic (e.g., if q and r then p), or a problem is goal-oriented (e.g., to satisfy p, satisfy q and r). Its application domain includes expert systems, natural language processing, symbolic algebra, VLSI circuit analysis, and relational databases. For the past several years, inductive logic programming has made significant progress for practical applications as a subdomain of machine learning [11, 3] (for more on Prolog, see [13]).

Since both of Prolog and fuzzy systems have wide areas of application, combining the two is a logical extension. Basically, potential application areas of such combined systems can be obtained by "fuzzifying" the Prolog application areas. They include fuzzy knowledge-based systems, such as fuzzy expert systems which may use fuzzy if–then rules, and fuzzy symbolic machine learning; fuzzy software engineering which may incorporate fuzziness in its programs and data; fuzzy databases which store and retrieve fuzzy information; and applications in medicine, economics and management which involve fuzzy information processing.

Because of its unique characteristics, Prolog requires special techniques for implementing fuzzy

0165-0114/98/$19.00
PII S0165-0114(96)00389-2

sets. Although various methods are discussed in the literature, each covers a relatively small number of techniques. This article presents a comprehensive overview of existing, as well as some new, methods of defining and manipulating fuzzy sets in Prolog. The article starts with a brief overview of fuzzy Prolog systems, followed by discussions on how to implement fuzzy sets in Prolog.

2. Fuzzy Prolog systems – a general overview

A common combination of Prolog and fuzzy systems is to extend Prolog, which is based on ordinary predicate logic, to the system based on fuzzy logic [2, 12]. Research work in fuzzy Prolog systems includes theoretical foundations and actual implementation of such systems. Also, applications of fuzzy Prolog to areas such as expert systems, databases, or specific fields such as medicine are practical, important issues.

There are two major ways of implementing fuzzy Prolog. One is to incorporate fuzzy representations and operations on top of an existing Prolog implementation [7, 9, 19, 21, 22]. We write a Prolog program that can, for example, perform fuzzy set or logic representations and operations. The second way is to develop a new, extended Prolog language that is based on fuzzy logic rather than first-order predicate logic [4–6, 8, 10, 18]. In this second approach, we have two implementation choices. We may develop our own interpreter using a meta-language, such as Prolog, Lisp, or C, or we may use a system already developed.

There are advantages and disadvantages to these approaches. An extended Prolog language might be more efficient and comprehensive than running a program written in ordinary Prolog. But a program written in such a dialect may not be easily transportable. If we develop our own extension, it may be time consuming. If we try to use a system developed elsewhere, it may be costly, may not be easily available, and/or it may be difficult to learn. However, such systems may have unique features and computation power. These advantages and disadvantages are just the opposite for the first approach – a program in Prolog with fuzzy representations. For example, Prolog interpreters/compilers are available on many systems and writing a short program on such systems can be done easily without much cost. Also, Prolog is often used for fast prototyping. Hence, the first approach may be simple yet good enough in many cases. Hereafter, this article will mainly discuss this first approach of implementing fuzzy sets in Prolog.

3. Implementing fuzzy sets in Prolog – a general note

3.1. Symbolic versus specific or numeric

For integral calculus, for example, we can numerically integrate a function for a specific domain, obtaining a specific numeric answer. This has been a common use of computers for scientific applications from the beginning of the use of computers. The other approach is symbolic integration of a function, e.g., integrating $\cos x$ giving $\sin x$ (plus a constant) without specific numeric values. Prolog is particularly powerful for this type of AI oriented symbolic manipulation of mathematical expressions, and this is the case in fuzzy set implementation in Prolog.

We note that each of these two types of computations, numeric and symbolic, has its own merits in general. For example, symbolically evaluating or simplifying a mathematical expression helps to understand better the characteristics of the solution. Normally, numeric answers hardly reveal their underlying characteristics. If necessary, the symbolic solution can further be used for more efficient numeric computation, whereas directly evaluating the original expression is less efficient. For example, numerically integrating $\cos x$ from 0 to π by Simpson's rule is more costly and less accurate than substituting the boundary values into the symbolically integrated answer $\sin x$. However, to obtain a specific output for an engineering system, for example, obviously we must have a specific numeric value, rather than a symbolic answer. This concept, symbolic versus specific or numeric, also applies to fuzzy sets. That is, fuzzy sets can be symbolically manipulated, or computed for specific sets yielding specific answers.

For symbolic manipulation of fuzzy sets, we can employ the technique used for other symbolic computations in Prolog. That is, we define the basic characteristics of each operator, such as the precedence, operator position relative to the operands, and

associativity, by using the built-in predicate "op" as, e.g.,

:-op(50, *yfx*, union).

We also define laws of operations on these operators. For example, one of the demorgan's laws can be specified in form of a Prolog rule as

conv(~(*A* intersection *B*), ~*A* union ~*B*):-!,

where conv is a user defined predicate in form of conv(*P*, *Q*) to convert *P* to *Q*, and "~" represents the complement.

Hereafter we concentrate on specific rather than symbolic manipulation of fuzzy sets. Also, we use the term "sets" to mean fuzzy sets unless noted otherwise.

3.2. Resolving differences between Prolog and fuzzy sets

Prolog, as in most other high-level languages, does not have sets or fuzzy sets as its built-in facility. Hence, we need to implement sets by using other built-in facilities such as lists or structures. This is analogous to implementing other common data structures such as stacks, queues, trees, and so on, when they are not part of the language facilities. To implement sets by another built-in facility, however, we have to adjust the differences between the two data structures. There are two most major differences between sets and common built-in facilities. One is that the order of elements in sets does not matter while it does in the built-in facilities. The second difference is that a set does not include duplicated elements (if there are, they are interpreted as representing the same element). Various set operations need special attention when nonset data structures are used.

The most common and easy solution to the second problem is to maintain the elements in each set to be unique. Each set operation must follow this condition. There are two methods to solve the first problem – the order of the elements in each set should not matter. First method is to follow the definition and to place the elements in any order. Subsequent operations usually require checking all the elements for a specific element in each set, since the element can be anywhere in the set. The second method is to impose an internal restriction for the set representation – the elements must be in order. When the universe is defined, the order of elements in each set is typically the same as in the universe. Note that we are not changing the basic rule of sets, we are merely imposing this restriction as a means of internal representation. There are advantages and disadvantages to these two methods regarding the time complexities of the operations. The second method requires more time to build sets unless the elements are sorted already, but less time for subsequent operations. A rule of thumb is that if sets do not change much and many operations follow, then use the second method.

3.3. Scaling degrees

An essential characteristics of fuzzy sets is to associate a degree with each element, most commonly a real number between 0 and 1. Since many Prolog systems have only integers, we need to represent real numbers by integers. This can be done by scaling the numbers, i.e., by multiplying a real number by a scaling constant k. For example, every real number can be multiplied by $k = 1000$, e.g., 0.5 can be represented by 500. In this representation, "linear" operations such as addition, subtraction, max, and min, need no readjustment, but "nonlinear" ones such as multiplication, division, and taking a square root, need rescaling. In the following we assume that such scaling is already taken care of, and use real number representation for easier reading.

4. Representing fuzzy sets in Prolog – specific methods

Fuzzy sets can be represented as extensions of ordinary sets in Prolog by associating the degrees or membership functions with the elements. Since there are many methods of representing ordinary sets in Prolog and several ways of associating the degrees, fuzzy sets can be implemented in many forms. Table 1 shows the categories of fuzzy set implementations discussed in this article.

4.1. Specific representation of ordinary sets

The two most common methods of explicitly representing ordinary sets are as follows.

Table 1
Categories of fuzzy set implementation in Prolog

I.	Construction of fuzzy set facilities on top of standard Prolog
Ia.	Symbolic
Ib.	Specific. Elements in each fuzzy set are described, either explicitly or implicitly
	Method 1. Explicit. A list for a fuzzy set (finite)
	Method 2. Explicit. A fact for every element (finite)
	Method 3. Implicit. Characteristics of elements are described (finite, infinite)
	Method 4. Other methods
II.	New extended Prolog languages that include fuzzy set facilities

Method 1: Explicit description of finite sets using lists. For example, ordinary set $\{a,b,c\}$ can be represented as $[a,b,c]$. Extensions of this representation are to associate a name of a set as, e.g., $s = \{a,b,c\}$, which can be represented as $s([a,b,c])$ or set$(s,[a,b,c])$. The predicate "set" is user-defined and can be any other name.

Method 2: Explicit description of finite sets using a fact for every element. For example, ordinary set $\{a,b,c\}$ can be represented as

data(a). data(b). data(c).

Or, set $s = \{a,b,c\}$ can be represented as

data(s,a). data(s,b). data(s,c).

The predicate "data" is user-defined and can be any other name.

The representation of Method 1 is more compact than that of Method 2. Also, Method 1 is more suitable to represent various types of sets such as the empty set [ ], a set of sets, e.g., [[2], [2, 3]]. However, there are situations in which the Method 2 type of set representation is particularly useful. One key point is that this representation is simply a collection of facts which are basic components of any Prolog program. Thus, in many cases operations on such a collection of facts may actually involve set operations even though we may not recognize them as such.

4.2. Associating degrees with elements

The above methods for ordinary sets can be extended to fuzzy sets by associating degrees with the elements. For Method 1, degrees can be associated with the elements as follows:

An element and its degree can be paired in some form. For example, to represent a fuzzy set of two elements, $\{a/0.5, b/1.0\}$, we can have

$[a/0.5, b/1.0]$, $[a, 0.5, b, 1.0]$, $[(a, 0.5), (b, 1.0)]$,

$[[a, 0.5]$, $[b, 1.0]]$, etc.

Similarly, Method 2 representation can be extended to fuzzy sets as, for example,

data$(a/0.5)$, data$(a, 0.5)$ or data$(s, a/0.5)$,

data$(s, a, 0.5)$, etc.

Prolog's pattern matching then can be used to access the degree of a specific element.

Another way of representing the degrees of set elements is to list them separately from the elements. For example, for the list representation of the elements, such as $[a,b]$, a separate list of the corresponding degrees can be made. Suppose that the universe $U = [a,b,c]$. A fuzzy set can be represented in list form, as e.g., [0.5, 1,0], or in structure form as e.g., degree (0.5, 1, 0), where 0.5, 1, and 0 are the degrees of elements a, b and c, respectively. This representation can be considered as an extension of the bit vector for an ordinary set, where its elements are extended from the domain of binary (0, 1) to real [0, 1]. Advantage and disadvantages of this "degree vector" method in comparison with the previous "paired" method of element degree are basically the same as in case of ordinary sets. That is, if $|U|$ is large, and the number of non-zero degree elements is small (i.e., sparse), this is not efficient. Otherwise, however, set operations are generally simpler.

4.3. Implicit representation of sets (Method 3)

A finite or infinite set can also be represented implicitly by describing the characteristics of the elements, rather than explicitly writing out all the elements (e.g., mathematically, $\{X \mid X$ is an integer and $1 < X < 10\}$). When a fuzzy set is represented in this implicit form, a formula for the degree, i.e., a membership function, has to be specified in the definition of the set. For example, suppose that a database is given as a collection of facts in form of

```
city(usa, new_york, 7100).
city(usa, washington, 600).
city(england, london, 6700).
```

We can define a set of large_cities implicitly as

```
city_in(large_cities, City, Degree):-
  city(_, City, Pop),
  ((Pop<3000, Degree is 0) | (Pop>7000,
    Degree is 1)| (Pop >=3000, Pop<= 7000,
    Degree is (Pop-3000)/4000)).
```

(In this article, "|" represents disjunction; in some systems ";" may have to be used in place of "|".) The corresponding explicit form would be

```
city_in(large_cities, new_york, 1).
city_in(large_cities, washington_dc, 0).
city_in(large_cities, london, 0.925).
```

5. Operations on fuzzy sets

5.1. Operations on Method 1 representation

Various fuzzy set operations can be defined similar to ordinary set operations. For example, procedure union $(S1, S2, S3)$ can be defined to compute the union of fuzzy sets, i.e., it succeeds if fuzzy set $S3$ is the union of fuzzy sets $S1$ and $S2$. There are several different definitions of fuzzy union, and the procedure union can be specified accordingly. The most common definition of fuzzy union is "max," namely max(degrees of corresponding elements in $S1$ and $S2) \rightarrow$(degree of corresponding elements in $S3$). Using this definition and representing two sets $S1$ and $S2$ as, e.g., $[a/0.5, b/1.0]$ and $[a/0.7, b/0.8]$, respectively, we would have $S3 = [a/0.7, b/1.0]$. The following is a simple implementation of the union operation, assuming all the elements in the universe are in both sets $S1$ and $S2$, possibly with zero degrees for some elements:

```
union([ ],[ ],[ ]):- !.
union([X/D1|T1],[X/D2|T2],[X/D3|T3]):-
  maxpair(X/D1,X/D2,X/D3), union(T1,T2,T3).
maxpair(X/D1,X/D2,X/D3):-
  D1>= D2,D3=D1,!.
maxpair(X/D1,X/D2,X/D2).
```

Under the same assumption, subset$(S1, S2)$, where $S1$ is a subset of $S2$, can be defined as

```
subset([ ],[ ]):- !.
subset([X/D1|T1],[X/D2|T2]):-
  D1 <= D2, subset(T1,T2).
```

For more details, see [9, 22].

5.2. Operations on Method 2 representation

When sets are represented by Method 2, i.e., using one fact for each element, two approaches are possible to perform set operations. One is to convert to Method I representation, then to perform Method 1 operations [13]. For example, data$(a/0.5)$, and data$(b/0.7)$ can be collected into a list form of $[a/0.5, b/0.7]$. Or, data$(s, a/0.5)$, and data$(s, b/0.7)$ can be collected into set$(s, [a/0.5, b/0.7])$. The second approach directly operates on facts. For example, suppose that two sets s and t are represented in form of data$(s, a/0.5)$, data$(s, b/0.7)$, data$(t, a/0.8)$, and data$(t, b/0)$. The following procedure generates the facts for the union U of sets $S1$ and $S2$:

```
union(S1,S2,U):-
  data(S1,X/D1), data(S2,X/D2),
  max(D1,D2,D),
  assertz(data(U,X/D)), fail |!.
max(D1,D2,D):- D1 >= D2,D = D1,!.
max(D1,D2,D2).
```

When this procedure is invoked by ?-union(s, t, u) for the above example, it will generate data$(u, a/0.8)$, data$(u, b/0.7)$. Incidentally, rewriting the first clause of max as max$(D1, D2, D1)$:- $D1 >= D2$, !. is a common error, since then ?-max$(5, 4, 4)$ will succeed with the last argument = 4.

5.3. Operations on Method 3 representation

From implicit representation of sets, we can generate explicit elements of sets for given characteristics.

Operations for Method 1 or 2 can then be applied. In the first step, all of the elements, or only necessary elements can be generated on demand.

6. Other methods

There are many other methods for representing and manipulating fuzzy sets that can be implemented in Prolog. A structure, rather than a list, can be used to represent a set, for example, $s(a/0.5, b/1.0, c/0.3)$. To access the elements, built-in procedures "arg" and "=.." (univ) can be used.

A tree using a structure in form of e.g., t(Root, Left_Subtree, Right_Subtree), in general, and $t(a/0.5, t(b/1.0, \text{nil}, \text{nil}), t(c/0.3, \text{nil}, \text{nil}))$ as a specific example, is another form of representation. Well organized tree representation, such as search tree, allows efficient search for elements.

7. Extensions

Fuzzy set representations can be extended to "higher-dimensional" mathematical structures such as Cartesian products and binary relations. They can be viewed as extensions from one-dimensional to two-dimensional sets. These data structures can be constructed by nesting the representations for sets, as e.g., lists of lists, structures of structures, etc.

8. Comparisons implementing fuzzy sets in Prolog and other languages

8.1. Lisp

Prolog and Lisp are the two major AI languages. There are many similarities between these languages. For example, both are symbolic, programs can be manipulated as data, they make extensive use of lists as data structures, and extensive use of recursion as a programming technique. Hence, many notes described in this article also apply to Lisp. They include implementation of fuzzy sets on top of Lisp or development of new extended languages [20], symbolic versus specific, resolving differences between Lisp and fuzzy sets, explicit representation of fuzzy sets by using lists and by associating degrees with elements, and implicit representation of fuzzy sets. Associated fuzzy set operations can also be defined as similar to the Prolog counterparts.

One common question is a comparison between Prolog and Lisp in general. As in the case of any languages (for example, C vs. Cobol, conventional vs. symbolic, etc.), a comparison is not easy since there are so many characteristics associated with the languages. For more on Prolog versus Lisp, see [13].

8.2. Conventional languages

Implementing fuzzy sets in conventional languages such as C is different from implementing fuzzy sets in symbolic languages such as Prolog. Symbolic manipulation of ordinary or fuzzy sets would be much harder in conventional than in symbolic languages. Representation and operation of fuzzy sets in explicit form can be implemented as extensions of ordinary sets by associating degrees with elements. Common representations include arrays, linked lists, and bit vectors [1].

In general, Prolog is often used for fast prototyping of a system. A successful Prolog system that is likely to be used extensively is sometimes rewritten in C for faster computation. In such a case, some form of a symbolic manipulation routine in C may be developed.

9. Conclusions

We have discussed various methods and their advantages and disadvantages for representing and manipulating fuzzy sets primarily in Prolog. The choice of a method depends on many factors, such as the type of applications, the type and size of the database, the characteristics of set operations performed, whether an implicit description of the elements is possible, the required computation time and storage space, and so on. As in other data structures and languages, the selection of methods will significantly affect programming and run time efficiency.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA, 1974).

[2] K. Atanassov and C. Georgiev, Intuitionistic fuzzy prolog, *Fuzzy Sets and Systems* **53** (1993) 121–128.

[3] I. Bratko and S. Muggleton, Applications of inductive logic programming, *Comm. ACM* **38** (1995) 65–70.

[4] L. Deyi and L. Dongbo, *A Fuzzy PROLOG Database System* (Research Studies Press, Wiley, New York, 1990).

[5] D. Dubois and H. Prade, A discussion of uncertainty handling in support logic programming, *Internat. J. Intelligent Systems* **5** (1990) 15–42.

[6] I. Graham, FRIL: a support logic programming system, *Expert Systems* **6** (1989) 186–190.

[7] C.J. Hinde, Fuzzy prolog, *Internat. J. Man-Machine Stud.* **24** (1986) 569–595.

[8] M. Ishizuka and N. Kanai, Prolog-ELF incorporating fuzzy logic, *New Generation Comput.* **3** (1985) 479–486.

[9] D.M. Kaminski, Introducing the fuzzy paradigm using prolog, *SIGCSE Bull.* **24** (1992) 202–206.

[10] T.P. Martin, J.F. Baldwin and B.W. Pilsworth, The implementation of FProlog – a fuzzy prolog interpreter, *Fuzzy Sets and Systems* **23** (1987) 119–129.

[11] S. Muggleton (Guest Ed.), Inductive logic programming, *SIGART Bull.* **5** (1994) 5–49.

[12] M. Mukaidono, Z. Shen and L. Ding, Fundamentals of fuzzy prolog, *Internat. J. Approximate Reasoning* **3** (1989) 179–193.

[13] T. Munakata, Notes on implementing sets in prolog, *Comm. ACM* **35** (1992) 112–120.

[14] T. Munakata, Practical AI: where it's been, and where it is now, *IEEE Expert* **8** (1993) 3–5.

[15] T. Munakata (Guest Ed.), Commercial and industrial AI, *Comm. ACM* **37** (1994) 23–119.

[16] T. Munakata (Guest Ed.), New horizons of commercial and industrial AI, *Comm. ACM* **38** (1995).

[17] T. Munakata and Y. Jani, Fuzzy systems: an overview, *Comm. ACM* **37** (1994) 69–76.

[18] I.P. Orci, Programming in possibilistic logic, *Internat. J. Expert Systems* **2** (1989) 79–96.

[19] B.L. Richards, When facts get fuzzy, *BYTE* **13** (1988) 285–290.

[20] Z.A. Sosnowski, FLISP – a language for processing fuzzy data, *Fuzzy Sets and Systems* **37** (1990) 23–32.

[21] D.B. Suits, Sometimes true and unequivocally indeterminate, *Comput. Language* **5** (1988) 39–43.

[22] Y.J. Tejwani and R.A. Jones, Decision support for fuzzy, probabilistic and control processes: a prolog assistant, *Proc. SPIE* **635** (1986) 394–399.