THE REPRESENTATION OF FUZZY KNOWLEDGE

by

Rick LeFaivre

DCS-TR-33

#44

Department of Computer Science
Rutgers University
The State University of New Jersey
New Brunswick, New Jersey 08903
January, 1974

ABSTRACT

A new AI programming language (called FUZZY) is introduced which provides a number of facilities for efficiently representing and manipulating fuzzy knowledge. A fuzzy associative net is maintained by the system, and procedures with associated "procedure demons" may be defined for the control of fuzzy processes. Such standard AI language features as a pattern-directed data access and procedure invocation mechanism and a backtrack control structure are also available.

This paper examines some general techniques for representing fuzzy knowledge in FUZZY, including the use of the associative net for the explicit representation of fuzzy sets and fuzzy relations, and the use of "deduce procedures" to implicitly define fuzzy sets, logical combinations of fuzzy sets, linguistic hedges, and fuzzy algorithms. The role of inference in a fuzzy environment is also discussed, and a technique for computing fuzzy inferences in FUZZY is examined.

The programming language FUZZY is implemented in LISP, and is currently running on a UNIVAC 1110 computer.

1. INTRODUCTION

There has been a great deal of research in the past few years concerned with formalisms for representing fuzzy knowledge (e.g., see Zadeh [7], [8], [9], and Goguen [3], [4]). Most of this research, however, has dealt with theoretical abstractions, and has ignored the issue of how fuzzy knowledge might be actually represented and utilized in a problem-solving environment. Such issues as how fuzzy information is stored, what facilities are available for manipulating fuzzy information, and how efficiently these manipulations may be carried out are of prime importance if fuzzy knowledge is to ever be effectively utilized in artificial intelligence.

This paper addresses the issue of how such abstract formalisms as fuzzy sets and fuzzy algorithms, and such ill-specified processes as fuzzy inference, can be efficiently represented in a problem-solving environment. The focus for the research described here is a new programming language (called FUZZY) which provides facilities for the efficient storage, retrieval, and manipulation of fuzzy knowledge.

2. A BRIEF INTRODUCTION TO FUZZY

FUZZY is a LISP-based "AI language" (Bobrow and Raphael [2]) which provides a number of facilities for representing and manipulating fuzzy knowledge, i.e., knowledge which is vague, imprecise, uncertain, ambiguous, inexact, or probabilistic in nature. An

early version of the language was described in [5], and a complete description of the system as it now stands may be found in [6]. We present here a brief overview of the language, and refer the reader to [6] for detailed descriptions of the operation of the various primitives.

Fuzzy Expressions

FUZZY is in essence a many-valued programming language. By this we mean that, unlike traditional languages in which expressions evaluate to a single value, FUZZY expressions may return both a value and a "fuzzy truth-value", or Z-value. (The term "Z-value" was chosen so as not to give any semantic connotations to the usage of this numeric modifier, since it can represent a conventional truth-value, a fuzzy grade of membership, a probability, a simple weight, or anything else the user wishes.) Internally, Z-values are stored by CONSing (appending) the Z-value to the value it modifies. Thus the construct

[(BIG BLOCK4) . 0.6]

represents a value of (BIG BLOCK4) and a Z-value of 0.6. Primitives are available for retrieving the value (VAL) or Z-value (ZVAL) portions of an expression, or for computing logical combinations of fuzzy expressions (ZAND, ZOR, ZNOT).

In addition to being able to return values and Z-values, FUZZY expressions may either succeed or fail in a manner similar to other AI languages (see [2]). Failure is caused by simply

returning a value of *FAIL, with any other value constituting success. Several variations of a standard IF-THEN-ELSE mechanism are available for controlling success and failure of individual expressions, and a user-controlled backtrack mechanism is available via the FOR statement.

The Associative Net

In addition to standard LISP data structures, FUZZY allows fuzzy knowledge to be explicitly represented by maintaining an associative network of assertions with optional Z-value modifiers. Standard ADD and REMOVE primitives are available for maintaining this net, while the FETCH primitive may be used to access the net via a powerful structural pattern-matcher. We shall examine the utilization of the associative net for the explicit representation of fuzzy knowledge in section 3 of this paper.

Fuzzy Procedures

It is often the case that knowledge is complex enough to prevent it from being stored explicitly in the form of simple assertions. FUZZY thus provides a powerful method of implicitly representing fuzzy knowledge via the specification of procedures (using the PROC statement). Associated with each procedure is a procedure demon which specifies the form of global control which is to be used to monitor the execution of the procedure. For example, the standard default demon causes a procedure to fail (i.e., return *FAIL) unless each of the expressions in the procedure

succeeds with a Z-value above a certain threshold. If this condition is met (or if the user explicitly causes the procedure to terminate successfully via the primitive SUCCEED), the minimum Z-value encountered is returned as the Z-value of the result. Other forms of global control can easily be specified by simply changing the demon associated with a particular procedure. We shall examine the applicability of this mechanism to the problem of fuzzy inference in section 6 of this paper.

Deductive Mechanisms

Like other AI languages (see [2]), FUZZY allows assertions to be implicitly defined via the specification of deduce procedures. The primitive DEDUCE is similar to FETCH, except that instead of searching for an assertion in the associative net, it searches for a procedure which allows the desired assertion to be computed. A GOAL primitive is also available which first performs a FETCH, utilizing DEDUCE only if the assertion is not present explicitly in the net. The pattern-directed procedure invocation mechanism of FUZZY is similar to that of other AI languages, except that FUZZY allows Z-value modifiers to be computed and returned along with assertions. We shall examine the application of FUZZY's deductive mechanisms to the implicit definition of fuzzy sets in section 4 of this paper.

3. EXPLICIT REPRESENTATIONS

The simplest method of representing fuzzy knowledge in FUZZY is via the associative net. Although fuzzy assertions may take the form of complex list structures if desired, the typical usage is to explicitly define fuzzy sets and fuzzy relations via simple assertions, with Z-value modifiers representing the grade of membership associated with each element. For example, we might explicitly define the fuzzy set YOUNG by listing the degree of youngness of each person under consideration:

```
(ADD (YOUNG TOM) 0.5)
(ADD (YOUNG DICK) 0.7)
(ADD (YOUNG HARRY) 0.85)
```

We could then retrieve (as a Z-value) the grade of membership of DICK in the fuzzy set of young people via:

```
(FETCH (YOUNG DICK))
```

It should be noted that the system automatically keeps the assertions in the net sorted by Z-value. To retrieve the name of the youngest person in the net, we thus need only enter

```
(FETCH (YOUNG ?WHO))
```

(The name, in this case HARRY, will be bound to the FUZZY variable ?WHO, and may be subsequently accessed via !WHO.) It is also possible to place a lower bound on the range of Z-values which will be accepted. For example,

```
(FETCH (YOUNG ?X) 0.75)
```

will fail unless an assertion with a Z-value of a least 0.75 can be found (the assertion with the highest such Z-value will be returned). More complex constraints may also be placed on the retrieval of assertions, as described in [6].

Note that fuzzy relations may be explicitly defined in the same way that fuzzy sets are. For example, the following statements might be used to define the fuzzy relation LIKES:

```
(ADD (TOM LIKES DICK) 0.3)
(ADD (TOM LIKES HARRY) 0.8)
(ADD (DICK LIKES TOM) 0.5)
(ADD (DICK LIKES HARRY) 0.9)
(ADD (HARRY LIKES TOM) 0.75)
(ADD (HARRY LIKES DICK) 0.4)
```

We could then retrieve the name of someone who likes someone else with a strength of at least 0.85 via

```
(FETCH (?WHO LIKES ?WHOM) 0.85)
```

The FOR statement may be used to iterate through the elements of particular fuzzy sets. For example,

```
(FOR FETCH: (YOUNG ?X) (PRINT !X))
```

prints the name of each young person (starting with the youngest person in the net). As a more complex example, the following statement finds two people who like each other with strengths of at least 0.7:

```
(FOR FETCH: (?P1 LIKES ?P2) ZVAL: 0.7
     (FETCH (!P2 LIKES !P1) 0.7)
     (EXIT))
```

If the net contains only the assertions added earlier, this expression will bind ?P1 to TOM and ?P2 to HARRY.

4. IMPLICIT REPRESENTATIONS

It is often desirable (or necessary) that fuzzy sets be defined implicitly rather than via explicit enumeration of the members of the set. For example, we might wish to define the fuzzy set of young people in terms of the less fuzzy concept of age. The following simple deduce procedure implicitly defines youngness as being inversely proportional to age (for persons under 100):

```
(ADD DEDUCE:
(PROC (YOUNG ?X)
     (FETCH (AGE !X ?AGE))
     (SUCCEED (YOUNG !X) (/ (- 100 !AGE) 100.0))))
```

If we then enter:

```
(ADD (AGE RICK 27))
(DEDUCE (YOUNG RICK))
```

a Z-value of 0.73 would be computed.

Note that other deduce procedures might be present which specify alternative definitions of youngness for use in the case when a person's age is not known explicitly. For example, the following procedure asserts that being young is the complement

of being old:

```
(ADD DEDUCE:
(PROC (YOUNG ?X)
      (ZNOT (GOAL (OLD !X)))))
```

If it is known that a particular person is old with a Z-value of 0.4, the above procedure would cause a grade of membership of 0.6 in the set of young people to be computed. Note that the degree of membership of an individual in the set of old people may itself be implicitly defined (perhaps via the color of his hair, number of wrinkles in his skin, etc.).

Logical operations on fuzzy sets may be defined in a natural manner using the above technique. For example, rather than assigning a specific name to the complement of a fuzzy set as was done above, we might wish to allow complementary fuzzy sets to be referenced via assertions of the form ((NOT YOUNG) SAM). This could be done by defining the following deduce procedure:

```
(ADD DEDUCE:
(PROC ((NOT ?N) ?X)
      (ZNOT (GOAL (!N !X)))))
```

If we then enter

```
(ADD (NICE SAM) 0.8)
(GOAL ((NOT NICE) SAM))
```

a Z-value of 0.2 would be computed, giving the grade of membership of SAM in the fuzzy set of non-nice people.

Similarly, the connective AND might be defined by:

```
(ADD DEDUCE:
(PROC ((?N1 AND ?N2) ?X)
      (ZAND (GOAL (!N1 !X))
            (GOAL (!N2 !X)))))
```

We could then request the grade of membership of TOM in the fuzzy set of not nice but strong people via:

```
(GOAL (((NOT NICE) AND STRONG) TOM))
```

In addition to logical connectives, linguistic hedges as described in [9] may also be defined implicitly via the use of deduce procedures. For example, the following procedure defines the hedge very:

```
(ADD DEDUCE:
(PROC ((VERY ?N) ?X)
      (GOAL (!N !X))
      (SUCCEED ((VERY !N) !X) (* (ZVAL) (ZVAL)))))
```

If we then enter:

```
(ADD (BIG BERTHA) 0.9)
(GOAL ((VERY BIG) BERTHA))
```

a Z-value of 0.81 will be returned, as desired.

5. FUZZY ALGORITHMS

Zadeh ([8],[9]) introduced the concept of a fuzzy algorithm, i.e., an algorithm which contains statements which are fuzzy in nature. Although we shall not make a detailed comparison between Zadeh's formulation of fuzzy algorithms and FUZZY procedures, it

should be noted that the facilities available in FUZZY for the manipulation of fuzzy concepts make it an interesting candidate language for the representation of fuzzy algorithms.

Several different classifications of fuzzy algorithms are discussed in [9], including definitional, generational, behavioral, and decisional algorithms. FUZZY procedures analogous to these various types of fuzzy algorithms can easily be devised. For example, the deduce procedure for "youngness" in the preceding section may be thought of as a fuzzy definitional algorithm. It implicitly defines the fuzzy set YOUNG in terms of the less-fuzzy concept of age.

Zadeh defines a fuzzy decisional algorithm as "a fuzzy algorithm which serves to provide an approximate description of a strategy or decision rule." As an example of a fuzzy decisional algorithm, consider the problem of whether I should marry a particular girl or not. My (chauvinistic) decisional rule might be that I will marry her if she is either very pretty, moderately pretty and moderately rich, or very rich. This could be expressed as a fuzzy algorithm (in Zadeh's terminology) as follows:

1) IF very pretty THEN marry her.
2) IF moderately pretty OR moderately rich THEN marry her.
3) IF very rich THEN marry her.
4) ELSE don't marry her.

We can represent this fuzzy decisional rule in FUZZY as a simple procedure:

```
(PROC NAME: MARRY  ?GIRL
      (IFANY [GOAL (PRETTY !GIRL) 0.9]
             [IFALL [GOAL (PRETTY !GIRL) 0.5]
                    [GOAL (RICH !GIRL) 0.5]]
             [GOAL (RICH !GIRL) 0.9]))
```

(MARRY JANE) will then succeed if Jane is marriageable material, and fail if she is not. Note that the procedural representation lays bare the precise nature of the various computations which make up the decisional rule.

With one minor modification, the above procedure can also be used as a generational algorithm as well as a decisional algorithm:

```
(PROC NAME: MARRY  ?GIRL
      (IFANY [GOAL (PRETTY !GIRL) 0.9]
             [FOR GOAL: (PRETTY !GIRL) ZVAL: 0.5
                  (GOAL (RICH !GIRL) 0.5)
                  (EXIT)]
             [GOAL (RICH !GIRL) 0.9]))
```

When passed a girl's name, this version simply checks her marriage qualifications as before. When passed a pattern, however, a marriageable girl is generated (i.e., her name is returned). Thus (MARRY ?) returns the most marriageable girl available based on the criteria specified in the above procedure. Note that in this case the order in which the various acceptable alternatives are given defines the priorities of the prospective suitor. The user of the above procedure evidently values prettiness above richness. A fortune-hunter, however, would try first to find a rich girl by reordering the conditions.

Other, more complex examples of the use of FUZZY procedures

to define fuzzy algorithms may be found in [6].

6. FUZZY INFERENCE

Classical and Fuzzy Detachment

Along with other AI languages, FUZZY provides facilities for automatically "deducing" information via the use of procedures. In non-fuzzy domains, the use of deduce procedures to represent deductive arguments is relatively straightforward. For example, the classic argument "all humans are mortal; Socrates is a human; therefore Socrates is mortal" may be succinctly represented by the FUZZY statements:

```
(ADD DEDUCE:
 (PROC (MORTAL ?X)
       (GOAL (HUMAN ?X))))
(ADD (HUMAN SOCRATES))
(DEDUCE (MORTAL SOCRATES))
```

This argument is an example of the rule of detachment, which is represented in classical logic by the theorem:

(A and (A implies B)) implies B

In other words, given that "A" is true and "A implies B" is true, we may deduce that B is true.

Consider, however, the role of detachment in a non-standard (e.g., many-valued) logic. Here the formulas "A" and "A implies B" may be only partially true -- how then are we to compute a truth-value for B? In essence, we would like to define a "detachment

operator" detach which provides us with at least a lower bound for the truth-value of B:

$$[A \text{ detach } (A \text{ implies } B)] \leq [B]$$

(where [P] denotes the turth-value of the position P).
A logical candidate for detach is the and operator, whose usage would be consistent with both the definition of detachment in classical logic and our intuitive notion of the meaning of detachment. Note, however, that there are a variety of different truth-value computations which can be associated with and, including the minimum of the individual truth-values as in fuzzy logic and the product as in probabilistic logic (see [6]). Our concern in this section is with the effect that variant definitions of detach have on fuzzy detachment, and how different rules of detachment can be represented in FUZZY.

A Paradox

As an example of the use of fuzzy detachment, consider how one might resolve the classical "paradox of the heap" discussed by Black [1] and Goguen [3]. Although this paradox can be stated in a variety of forms, perhaps the simplest statement is the following: "1 is a small integer; if n is a small integer, then n+1 is a small integer; therefore (by induction) any integer N is small." Given a particular N, the proof that N is small consists of N-1 applications of detachment, starting with the fact that 1 is small and the implication that if 1 is small then 2 is small. The paradoxical nature of this argument is based

on the fact that starting from two quite reasonable premises (that 1 is a small integer and that the successor of a small integer is still small), we are able to arrive at a conclusion (that all integers are small) which contradicts our intuitive understanding of "smallness". The problem, of course, is that the set of "small integers" is not sharply defined -- we have attempted to impose the laws of classical logic on a non-classical (i.e., fuzzy) problem.

Resolution of the Paradox

We can make use of fuzzy detachment to resolve the above paradox by noting that the result of adding 1 to a small integer, while still small, is not quite as small as it was before. In other words, the statement "if n is small then n+1 is small" is not quite true -- let us say that it has a truth-value of 0.99. If we assume that the truth-value of the statement "1 is small" is 1.0, and we use multiplication for our detachment operator, it is trivial to show that the truth-value of the statement "N is small" is 0.99^{N-1}, which is an intuitively satisfying result. Hence, the "paradox" of classical logic becomes a valid deduction when formulated in many-valued logic.

The selection of multiplication for the truth-value computation in fuzzy detachment leads to the interesting consequence that the validity of a chain of nearly valid implications decreases as the length of the chain increases. It is this property which led to the resolution of the above paradox -- each

single application of the rule of detachment was quite valid; it was only as the chain of implications increased in length that the validity lessened. Note that if the minimum is used as the detachment operator, validity does not decrease as the length of the deductive chain increases. In this case, the validity of the entire chain is bounded by the least valid implication in the chain (i.e., the chain is as weak as its weakest link). The minimum, then, would not have been a viable detachment operator for use in the resolution of the paradox. There may be other problem areas, however, for which the minimum is more suitable than multiplication. Clearly, a fuzzy problem-solving system must allow for variant definitions of fuzzy detachment for use in different problem domains.

Detachment in FUZZY

Let us briefly examine the issue of fuzzy detachment in FUZZY by showing how the above solution of the paradox might be represented. We first define a procedure demon DETACH which represents the (multiplicative) detachment operator:

```
(CSETQ DETACH
(LAMBDA (V TH AC)
        (COND [<EQ V *FAIL> <FAIL>]
              [<EQ V *DONE>  AC]
              [T <* TH (ZVAL V)>])))
```

(The structure of procedure demons is discussed more fully in [6] -- the important point to note about the simple demon listed above is that it uses multiplication to compute a Z-value for the

procedure rather than simply returning the minimum value encountered.) We then add the two premises to the data base. As usual, the implication "if n is small then n+1 is small" is expressed as a deduce procedure which utilizes backwards chaining:

```
(ADD (SMALL 1))
(ADD DEDUCE:
 (PROC DEMON: DETACH  THRESH: 0.99  (SMALL ?N)
       (GOAL (SMALL &(SUB1 !N)))))
```

(The prefix "&" acts as an evaluation operator.) A request like:

```
(DEDUCE (SMALL 13))
```

would then cause the proper Z-value (0.99^{12}) to be computed.

Note that all of the Z-value manipulations are carried out by the procedure demon; once the demon is written, the user need only specify the threshold value, which in this case is analogous to the truth-value of the deduce procedure. The procedure demon and threshold value associated with a procedure may be changed if desired, leading to the interesting possibility of learning what form of detachment works best for a given problem domain.

7. CONCLUDING REMARKS

In this paper we have briefly examined some general techniques for representing fuzzy knowledge using the programming language FUZZY. We showed how fuzzy sets and relations can be stored and accessed explicitly using the associative net; how fuzzy sets may be defined implicitly via the use of deduce procedures; how

logical combinations of fuzzy sets and linguistic hedges can be implicitly defined; and how fuzzy algorithms and fuzzy inference mechanisms can be represented in FUZZY. The application of these various techniques to the solution of problems in such areas as deductive problem-solving, pattern classification and scene description, learning, and natural language processing is examined in [6].

The development of efficient, computationally specific formalisms for the representation and manipulation of fuzzy knowledge is an important goal if we are to ever effectively utilize fuzzy information in a problem-solving environment. The combination of the automatic deductive facilities present in modern AI languages with the ability to efficiently access and manipulate fuzzy knowledge (stored in both assertional and procedural form) makes FUZZY a promising vehicle for future research into the nature of fuzzy problem-solving.

REFERENCES

[1] Black, M.: "Reasoning with Loose Concepts". Dialogue, 2, 1-12 (1963).

[2] Bobrow, D. and B. Raphael: "New Programming Languages for AI Research". Technical Note 82, SRI AI Center (1973).

[3] Goguen, J.: "The Logic of Inexact Concepts". Synthese, 19, 325-373 (1969).

[4] Goguen, J.: "Axioms, Extensions and Applications for Fuzzy Sets: Languages and the Representation of Concepts". IBM Research Report RC 4547 (1973).

[5] LeFaivre, R.: "FUZZY: A Programming Language for Fuzzy Problem-Solving". Technical Report #202, Computer Sciences Dept., University of Wisconsin (1974).

[6] LeFaivre, R.: "Fuzzy Problem-Solving". Ph.D. Dissertation, Computer Sciences Dept., University of Wisconsin (1974). Available as Technical Report #37, Madison Academic Computing Center, University of Wisconsin.

[7] Zadeh, L.: "Fuzzy Sets". Information and Control, 8, 338-353 (1965).

[8] Zadeh, L.: "Fuzzy Algorithms". Information and Control, 12, 94-102 (1968).

[9] Zadeh, L.: "Outline of a New Approach to the Analysis of Complex Systems and Decision Processes". IEEE Trans. on Systems, Man, and Cybernetics, SMC-3, 28-44 (1973).