Extracting Fuzzy Rules Under Uncertainty and Measuring Definability Using Rough Sets

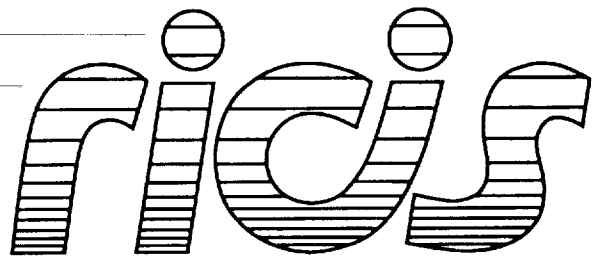
Donald E. Culas

University of Houston-Downtown

December 9, 1991

Cooperative Agreement NCC 9-16
Research Activity No. SR.01

NASA Johnson Space Center
Information Systems Directorate
Information Technology Division



Research Institute for Computing and Information Systems
University of Houston-Clear Lake

TECHNICAL REPORT

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information Systems (RICIS) in 1986 to encourage the NASA Johnson Space Center (JSC) and local industry to actively support research in the computing and information sciences. As part of this endeavor, UHCL proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a continuing cooperative agreement with UHCL beginning in May 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The UHCL/RICIS mission is to conduct, coordinate, and disseminate research and professional level education in computing and information systems to serve the needs of the government, industry, community and academia. RICIS combines resources of UHCL and its gateway affiliates to research and develop materials, prototypes and publications on topics of mutual interest to its sponsors and researchers. Within UHCL, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business and Public Administration, Education, Human Sciences and Humanities, and Natural and Applied Sciences. RICIS also collaborates with industry in a companion program. This program is focused on serving the research and advanced development needs of industry.

Moreover, UHCL established relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research. For example, UHCL has entered into a special partnership with Texas A&M University to help oversee RICIS research and education programs, while other research organizations are involved via the "gateway" concept.

A major role of RICIS then is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. RICIS, working jointly with its sponsors, advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research and integrates technical results into the goals of UHCL, NASA/JSC and industry.

Extracting Fuzzy Rules Under Uncertainty and Measuring Definability Using Rough Sets

RICIS Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Donald E. Culas of the University of Houston - Downtown. Dr. Andre' de Korvin was the UH - Downtown faculty advisor. Dr. A. Glen Houston served as the RICIS research coordinator.

Funding was provided by the Information Systems Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between the NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA technical monitor for this activity was Dr. Robert Savely of the Information Technology Division, NASA/JSC.

EXTRACTING FUZZY RULES UNDER UNCERTAINTY
AND
MEASURING DEFINIBILITY USING ROUGH SETS

Prepared by

Donald E. Culas

in

Partial fulfillment

of the requirements

for

CS 4395

Department of Applied Mathematical Sciences

University of Houston - Downtown

December 9, 1991

Committee Members / Approval

Dr.	Andre' de Korvin	Faculty Advisor	Andre de Korvin 12/9/91
Dr.	Kenneth Oberhoff	Member	
Dr.	Robert Lea	Member	
Dr.	Ananda Gunawardena	Member	
Dr.	Richard A. Alo'	Chairman	12/16/91

Abstract

Although computers have come a long way since their invention, they are basically able to handle only crisp values at the hardware level. Unfortunately, the world we live in consists of problems which fail to fall into this category, i.e. uncertainty is all too common. In this work, we look at a problem which involves uncertainty. To be specific we deal with attributes which are fuzzy sets. Under this condition we acquire knowledge by looking at examples. In each example a condition as well as a decision is made available. Based on the examples given to us, we will extract two sets of rules namely: certain and possible. Furthermore we will construct measures of how much we believe these rules, and finally we will define the decisions as a function of the terms used in the conditions.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I would like to thank Dr. Andre' de Korvin for letting me work on an interesting project such as this and for giving me his time and attention. Furthermore let me take this opportunity to thank Dr. Kenneth Oberhoff for his valuable suggestions and comments given to me during the software design and development stages. Finally, I would like to thank Dr. Richard A. Alo' for giving me an opportunity to take part in such a project, and for all the constructive comments given to me throughout the semester.

CHAPTER 1

INTRODUCTION

Despite the advancements made in computer technology to date, most of the computers on the market today are stored program sequential processing machines built around the Von Neumann architecture, the principles for which date back to the Turing machine, a computing model first proposed by Alan Turing in 1936. This means that, in principle, modern day computers are designed primarily to carry out mathematical calculations. Human expectations vis-a-vis computers know no bounds. These expectations go beyond routine jobs such as numerical calculations and the processing of office work, to include support in decision-making processes, the ability to understand natural languages, the diagnosis of malfunctions and the processing of intellectual information such as that required in design and planning work. To accomplish these kinds of operations, symbol processing computers equipped with inference functions are required. However, even symbol processing machines are not capable of handling experience and intuition, two very important aspects of intelligence. This is because conventional computers are extremely **crisp** . (i.e. capable of dealing with definite values) having being designed around the binary logic of Boolean algebra. Human experience and intuition, however, by their very nature are multi-numerical. In other words, they are **fuzzy**.

This raises the question of just how necessary it is to have computers that are capable of processing ambiguous information, such as human experience and intuition. After all can't most phenomena encountered in this world be thoroughly processed mathematically? This question has been raised because today's computers are only used to solve well-structured problems for which all information

is available. However, the everyday real world in which we live is rife with problems for which not all information is available, and which are not well structured. This project focuses on such a problem, for which all information is not available and/or not well structured.

1.1 The Problem at Hand

Expert systems of a certain kind rely essentially upon the availability of a method for handling uncertainty. These systems cannot be conceived without a decision being firstly made about the choice of this method. Obviously this is true for all expert systems using empirical knowledge which in itself is not absolutely certain. As an example, we could mention a medical expert system which draws conclusions from the observed symptoms about whether or not a certain disease is present. All conclusions of this type inevitably contain an amount of uncertainty. However the rules which lead to these conclusions should not be confused with classical logical rules and must not be treated in the same way.

We shall call expert systems of this type **diagnostic systems** . They are mostly in the field of medicine, but can also be used in many other applications such as meteorology or geology, and of course for the control of technical installations. Therefore the expression **diagnostic system** should always be understood in the sense of an expert system, which relies upon empirical interdependencies for drawing its conclusions and consequently requires the treatment of uncertainty.

In order to make it possible to decide upon an appropriate therapy, a quantitative measure of uncertainty has to be applied in all relevant cases of a diagnostic system. Moreover it may be sensible to establish rules which, in certain stages of the investigation, direct the investigator's efforts depending on the degree of certainty achieved for possible hypothesis.

It is evident therefore, that for researchers who design diagnostic systems the question has to answered, as to which method of measuring uncertainty should be employed. For more than three hundred years scientists, philosophers, mathematicians and statisticians have used the concept of probability to describe degrees of uncertainty. Over three centuries a huge amount of theoretical results and experiences concerning the applicability of probability theory in different fields of human knowledge has been accumulated. Nevertheless many doubts concerning the appropriateness of the use of probability in diagnostic systems have arisen during the last decade. In the following sections we look at the probability theory and see why this is so.

1.2 Reasoning and Probability Theory.

Decision making often involves the use of rules. Simple rules are acceptable to most people in their everyday life, e.g.

In India, if you are under 60 years of age then you are entitled to a retirement position.

A rule for entitlement might be more complex, but understandable, e.g.

If you are at least 60 years old and female and you have been resident in India for at least 25 years, or if you are male and at least 65 years old and you have been resident in India for at least 30 years then, provided you are not receiving a disability pension, you are entitled to the retirement position.

People use 'if .... then....' statements in conversation, and often use rules in their everyday lives. However, problems which require expertise are not deterministic, i.e. the solutions cannot be stated in simple rules. Where judgement is involved, people often use words like probably, unlikely, almost certainly, i.e. uncertainty is involved. In some cases they quantify what they mean. For example:

I am 99% confident that if you water the plant, its condition will improve.

There is a small risk, about 5%, that you have this disease.

The ways in which people use these percentages are ill defined and often inconsistent. However, as we shall see in subsequent sections, there are mathematical theories which provide logical models for uncertainty.

1.2.1 Probability Theory.

Probability theory originated in the seventeenth century in the context of gambling. A gambler assesses his chance of winning and therefore the risk associated with his bid [3]. This process is very similar to that of an expert weighing up evidence, and judging whether he has sufficient evidence to justify a particular course of action. Chance, expectation and risk are components of both probability theory and expert judgements.

Probability is a measure of certainty between 0 and 1. The extreme values denote impossibility and certainty. Most people would understand that if a fair coin is tossed then the probability of its landing on a certain side is 0.5. This is because we ignore the possibility of its landing on its edge or not landing at all, and the other two outcomes are equally likely. Furthermore, only one of the events (head or tail) can occur at once, i.e. the events are **mutually exclusive.** This leads us to the classical definition of probability:

> If a random experiment has N possible outcomes which are all equally likely and mutually exclusive, and n of these possibilities has outcome A then the probability of outcome A is n/N.

For example, consider a standard pack of 52 playing cards which has been shuffled so that the order of the cards is unpredictable. If a card is picked at random then the chance that it is a club is 13/52 = 0.25. This is a very simplistic view of uncertainty. The definition depends on the terms random, mutually exclusive and equally likely. It cannot help much with questions like:

What is the probability that a child born in the United States will be a male? What is the probability that the pain is caused by indigestion, and not a serious illness?

These are all real questions, and experts continually make similar judgements. If we looked at the record of births in the United States over the past two years then we could calculate the relative frequency of male births, i.e. the ratio of number of boys to number of births. We would expect this to be close to the true probability. Assuming that there had been no genetic changes, a more reliable estimate could be obtained from the records of the past ten years. So, if we can imagine a series of observations under constant conditions then the probability p of event A can be approximated by the relative frequency of A in a series of such observations. In practice 'true' probabilities are almost impossible to quantify, and most probabilities used are estimates based on relative frequencies.

1.3 Why not Classical Probability Theory?

Even though, the basic ideas prevailing in some considerations about diagnostic systems sound convincing, they violate fundamental requirements for reasonable handling of uncertainty. These ideas may be described as follows: If a certain fact is observed, a measure M_1 of uncertainty concerning the hypothesis in question must exist. If in addition another fact is observed, which produces a measure M_2 with respect to the same hypothesis, a combination rule must be given, which yields the measure of uncertainty of this hypothesis resulting from both observations. Such a rule, which calculates the measure of uncertainty for the combined observation as a function of the measures M_1 and M_2 can never take into account the kind of mutual dependence of the two observed facts. It might well be that these facts nearly always occur together, if indeed they occur at all. In such a situation the second observation is redundant and should not be used to update the

measure of uncertainty. In another situation the two facts very seldom occur simultaneously and if they do, then this is an important indication concerning the hypothesis in question. If they do occur simultaneously, the updating of the measure of uncertainty should have drastic consequences. Classical probability theory which treats these two situations equally cannot be considered useful.

Another argument against the probability theory is that : Is it justifiable to attribute a certain measure of uncertainty to the observation of a given fact, irrespective of the circumstances? For example let's take the example of a medical diagnostic system: If a symptom Z is observed, and a measure of uncertainty is used concerning the hypothesis of the presence of a certain disease, can this measure remain valid, if this disease occurs much more frequently than before? Once again an appropriate use of probability theory reveals the kind of dependence prevailing in this case. However, this will not be a popular result, because it states that a diagnostic system using this type of measure of uncertainty cannot be applied to populations showing different frequencies of this disease.

The problems of using probability models are compounded by the fact that people do not really understand the theory [6]. The theory itself is consistent and correct, but in order to apply it we need to make assumptions about underlying distributions and independence and sometimes use sophisticated mathematics to develop a consistent model for the system. Even given a consistent model people find it hard to estimate conditional probabilities, if enough data is not provided. Statistical tests are a method of using probability theory to judge the weight of evidence and of selecting an hypothesis from two alternatives. However, many of the theorems and methods needed when using probabilities in diagnostic systems require the expert to estimate probabilities, sometimes without recourse to relative frequencies. Yet another problem with forcing experts to describe their inference in terms of probability theory is that the theory is not a natural method of reasoning.

1.4 Discussion

Recently, a lot of time and effort has been expended by the expert systems research community to the acquisition of knowledge under uncertainty. Uncertainty arises in many different situations. It may be caused by the ambiguity in the terms used to describe a specific situation, it might be caused by skepticism of rules used to describe a course of action or by missing and/or erroneous data.

In order to deal with uncertainty, techniques other than classical logic need to be developed. Statistics is the best tool available to handle likelihood. However, in many cases probabilities need to be estimated, sometimes without even recourse to relative frequencies. Estimates, then are likely to be very inaccurate. Many authors have cited theoretical weaknesses of expert systems based on statistical technique. In particular, there has been an attempt to create a system for the verification of indications for treatment of duodenal ulcers by HSV on the basis of statistics. The results were counter-intuitive and the system was rejected by physicians. The Dempster- Shafer theory of evidence or the theory of belief functions, give a useful measure for the evaluation of subjective certainty. The Dempster-Shafer theory has recently become popular. For an in depth-look at the Dempster-Shafer theory the reader is referred to [10]. Fuzzy logic, based on Zadeh's theory of **fuzzy sets** (where the degree to which an optional element (a) belongs to set (A) is determined by assigning it a value or grade ranging from 0 to 1) is another means of handling uncertainty. However, this too has problems [9]. There is extensive literature on ways to deal with uncertainty in expert systems, like a combination of statistics and fuzzy logic, theory of endorsements [1], non-monotic logic [7, 8], modal logic etc. [5].

One of the most popular ways to acquire knowledge is based on learning from examples. An effective tool to infer knowledge from examples is **rough set**

theory. Rough set theory was introduced in 1981 by Z. Pawlak as a method to acquire knowledge under uncertainty. The main assumption of the rough set theory is that the information stored in the data base like system, called an information system, may contain inconsistencies. In the process of acquiring knowledge these inconsistencies are taken into account. Thus, using the basic tools of rough set theory, which we will look at closely in the next chapter, two sets of rules are produced namely **certain** and **possible**. The main advantage of the rough set theory is that it does not need any preliminary or additional information about data like probability in statistics. Moreover rough set theory has been successfully implemented in knowledge-based systems in medicine and industry. In particular, an expert system based on rough set theory for engineering design is being developed at Wayne State University, Michigan, and University of Regina, Canada.

1.5 Scope.

In this work, we will deal with a setting where a decision maker is faced with uncertain (i.e. fuzzy) symptoms and makes a fuzzy diagnosis which might be strongly or weakly based on these symptoms. The cases which we will look at are not "textbook cases" and the values of attributes are not crisp. Moreover the diagnosis is not of a "pure type". It is a mixture of several "pure types". Thus, a patient might have a diagnosis of the type $.3/D_A + .6/D_B$ meaning that the physician believes the fuzzy symptoms reflect disease D_A with strength .3 and disease D_B with strength .6. From such a setting we will extract fuzzy rules using the rough set theory.

Fuzzy rules are naturally present in descriptions, crisp rules are the exceptions. Also, fewer fuzzy rules are needed than crisp ones to build an expert system. Thus a rule such as : If the tumor is somewhat large then the presence of

skin cancer is somewhat likely is the type of rule experts naturally use as opposed to giving the size of a tumor and a number expressing the probability of cancer.

In the first part of this work we will develop a methodology to extract rules such as the ones stated above, from fuzzy symptoms and fuzzy diagnosis. In fact we will extract two sets of rules i.e. **certain** and **possible** rules as well as a measure of how much we believe these rules. In the second part we will look at a related problem that is to define the diagnosis in terms of the symptoms. In the next chapter we take an in-depth look at the rough set theory which is necessary to understand the rest of this paper.

CHAPTER 2

ROUGH SETS

Acquiring knowledge under uncertainty is one of the main problems of expert systems. One of the most popular ways to acquire knowledge is based on learning from examples. In 1981, Z. Pawlak introduced a new tool, namely rough set theory to acquire knowledge under uncertainty. In this chapter we look at the basic concepts of rough set theory. Other methods have been developed prior to the introduction of rough set theory. However, use of the rough set theory seem to have many advantages over the other methods. One of the main advantages of the rough set theory is that it does not need any preliminary or additional information about data (like probability in statistics, basic probability number in Dempster-Shafer theory, grade of membership, or the value of possibility in fuzzy set theory). Another advantage of rough set theory is that its algorithms are very simple, and the theory itself is clear and easy to follow. Moreover the theory has been successfully implemented in many cases in expert systems in medicine and industry.

2.1 Basic Notations and Concepts

All the concepts mentioned in this section can be found in [4]. Let U be a non empty set, called the **universe**, and let R be an equivalence relation on U called an **indiscernibility** relation. An ordered pair A = (U,R) is called an **approximation space.** For an element x of U, the equivalence class of R containing x will be denoted by $[x]_R$. Equivalence classes of R are called **elementary** sets in A. We assume that the empty set is also elementary. Any finite union of elementary sets in A is called a **definable** set in A.

Let X be a subset of U and we wish to define X in terms of definable sets in A. Thus, we need two more concepts. A lower approximation of X in A, denoted by $\underline{R}X$, is the set given by

$$\{x \in U \mid [x]_R \subseteq X\}.$$

An upper approximation of X in A, denoted by $\overline{R}X$, is the set given by

$$\{x \in U \mid [x]_R \cap X \neq \phi\}$$

The lower approximation of X in A is the greatest definable set in A, contained in X. The upper approximation of X is the least definable set in A containing X. A rough set in A (or rough set, if A is known) is the family of all subsets of U having the same lower and upper approximations in A.

Let X and Y be subsets of U. Lower and upper approximations of X and Y in A have the following properties:

$$\underline{R}X \subseteq X \subseteq \overline{R}X,$$

$$\underline{R}U = U = \overline{R}U,$$

$$\underline{R}\phi = \phi = \overline{R}\phi,$$

$$R(X \cup Y) \supseteq \underline{R}X \cup \underline{R}Y,$$

$$\overline{R}(X \cup Y) = \overline{R}X \cup \overline{R}Y,$$

$$\underline{R}(X \cap Y) = \underline{R}X \cap \underline{R}Y,$$

$$\overline{R}(X \cap Y) \subseteq \overline{R}X \cap \overline{R}Y,$$

$$\underline{R}(X - Y) \subseteq \underline{R}X - \underline{R}Y,$$

$$\overline{R}(X - Y) \supseteq \overline{R}X - \overline{R}Y,$$

$$\underline{R}(-X) = -\overline{R}X,$$

$$\overline{R}(-X) = -\underline{R}X,$$

$$\underline{R}X \cup \overline{R}(-X) = X,$$

$$\underline{R}(\underline{R}X) = \overline{R}(\underline{R}X) = \underline{R}X,$$

$$\overline{R}(\overline{R}X) = \underline{R}(\overline{R}X) = \overline{R}X,$$

where -X denotes the complement U-X of X.

Let x be an element of U. We define two additional membership relations $\underline{\in}$ and $\overline{\in}$, called **strong** and **weak memberships**, in the following way

$$x \underline{\in} X \text{ iff } x \in \underline{R}X$$

and

$$x \overline{\in} X \text{ iff } x \in \overline{R}X$$

with meanings: x is **certainly** in X and x is **possibly** in X respectively. Our terminology originates in that we want to decide if x is in X on the basis of definable sets in A rather than on the basis of X. This means that we deal with $\underline{R}X$ and $\overline{R}X$ instead of X, and since $\underline{R}X \subseteq X \subseteq \overline{R}X$, if x is in $\underline{R}X$ it is certainly in X. On the other hand, if x is in $\overline{R}X$, it is possibly in X.

2.2 Information Systems

An information system is similar to a data base. The difference is that the entities of such an information system, called objects, do not need to be distinguished by attributes. The information system serves as the basis for knowledge acquisition, producing rules from examples. Therefore, attributes are divided into two types: conditions and decisions (or actions). Objects are described by values of conditions, while classifications made by experts are represented by values of decisions.

For example, if the system is a hospital, the objects would be patients, the condition attributes would be tests, and the decision attributes would be diseases. Each patient would be characterized by test results and would be classified by physicians (experts) as being on some level of disease severity. As another example if the system is an industrial process, the objects would be sample of processes taken at some specific moments in time. Conditions would be the parameters of the process, while the decisions would be actions taken by the operators (experts).

An information system S is a quadrapule (U,Q,V,P)where U is a non empty finite set, and its elements are called objects of S, $Q = C \cup D$ is a set of attributes, C is a non empty finite set, its elements are called condition attributes of S, and D is also a non empty finite set, and its elements are called decision attributes of S, $D \cap C = \phi$.

$V = \bigcup_{q \in Q} V_q$ is a non empty finite set, and its elements are called values of attributes, where V_q is the set of values of attribute q, called the domain of q, and p is a function of $U \times Q$ onto V, called a description of S, such that $p(x,q) \in V_q$ for all $x \in U$ and $q \in Q$

Let P be a nonempty subset of Q, and let x, y be members of U. Objects x and y are indiscernible by P in S, denoted by $x \tilde{P} y$, iff for each q in P, p(x,q) = p(y,q). Obviously, $\tilde{P}$ is an equivalence relation on U. Thus P defines a partition on U; such a partition is a set of all equivalence classes of $\tilde{P}$. This partition is called a classification of U generated by P in S, or briefly a classification generated by P.

2.3 Rough Definibility of a Set

For a non empty subset P of Q, an ordered pair $(U, \tilde{P})$ is an approximation space A. For the sake of convenience, for any $X \subseteq U$, the lower approximation of X in A and the upper approximation of X in A will be called P-lower approximation of X in S and P-upper approximation of X in S, and will be denoted by $\underline{P}X$ and $\overline{P}X$, respectively. A definable set X in A will be also called P-definable in S. Thus, X is P-definable in S iff $\underline{P}X = \overline{P}X$

For a non empty subset P of Q, a set $X \subseteq U$ which is not P-definable in S = (U, Q, V, P) will be called P-undefinable in S. Set X is P-undefinable iff $\underline{P}X \neq \overline{P}X$

The set X will be called roughly P-definable in S iff $\underline{P}X \neq \phi$ and $\overline{P}X \neq U$.

The set X will be called internally P-undefinable in S iff $\underline{P}X = \phi$ and $\overline{P}X \neq U$.

The set X will be called externally P-undefinable in S iff $\underline{P}X \neq \phi$ and $\overline{P}X = U$.

The set X will be called totally P-undefinable in S iff $\underline{P}X = \phi$ and $\overline{P}X = U$.

For an internally P-undefinable set X in S we can not say with certainty that any $x \in U$ is a member of X. For an externally P-undefinable set X in S we cannot exclude any $x \in U$ being possibly a member of X. In the next section we look at an example which illustrates the above mentioned concepts.

2.4 An Example

Let us look at the information system which is given by the following table.

Table 1. An example of information system:

	C		
	C		D
	Temperature	Headache	Influenza
X1	normal	no	no
X2	normal	yes	no
X3	normal	yes	yes
X4	subfebrile	no	no
X5	subfrebile	yes	no
X6	subfebrile	yes	yes
X7	high	no	yes
X8	high	yes	yes
X9	high	yes	yes

The classification, generated by the set C of conditions attributes, called **Temperature** and **Headache**, is equal to

$$\{ \{X_1\}, \{X_2, X_3\}, \{X_4\}, \{X_5, X_6\}, \{X_7\}, \{X_8, X_9\} \}.$$

The set D of decision attributes consists of one member, called **Influenza**. As can be seen in the table, an expert introduced two inconsistencies. First, he assigned different values of condition attributes to patients x_2 and x_3, in spite of the fact that both patients, x_2 and x_3, characterized by the same values of condition attributes **Temperature** and **Headache**. Yet another inconsistency is associated with patients x_5 and x_6.

Let us assume that $X = \{ x \mid p(x, d) = no \}$, i.e. $X = \{x_1, x_2, x_4, x_5\}$. Thus X represents all patients in U, classified by an expert in the same way, as being not sick with influenza. Then

$$\underline{C}X = \{x_1\} \cup \{x_4\} = \{x_1, x_4\},$$

$$\overline{C}X = \{x_1\} \cup \{x_2, x_3\} \cup \{x_4\} \cup \{x_5, x_6\} = \{x_1, x_2, x_3, x_4, x_5, x_6\}$$

It is the presence of inconsistencies that produce a difference between the lower and upper approximation.

In our example, $\underline{C}X \neq \phi$ and $\overline{C}X \neq U$, therefore X is roughly C-definable in S. For set X, sets $\underline{C}X$ and $\overline{C}X$ are illustrated by the following figures:

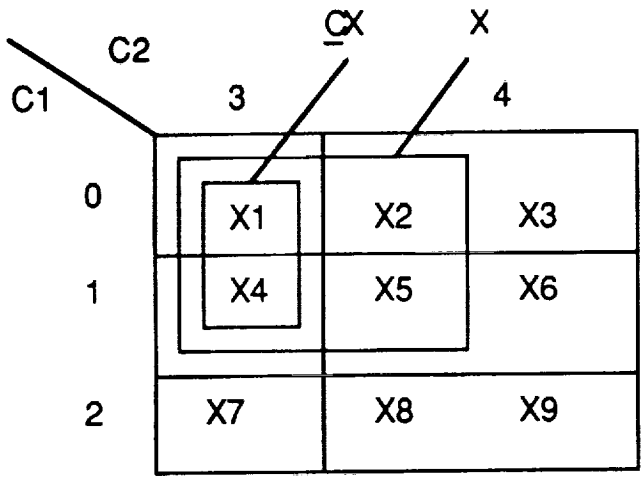


Figure (a) lower approximation $\underline{C}X$ of set X

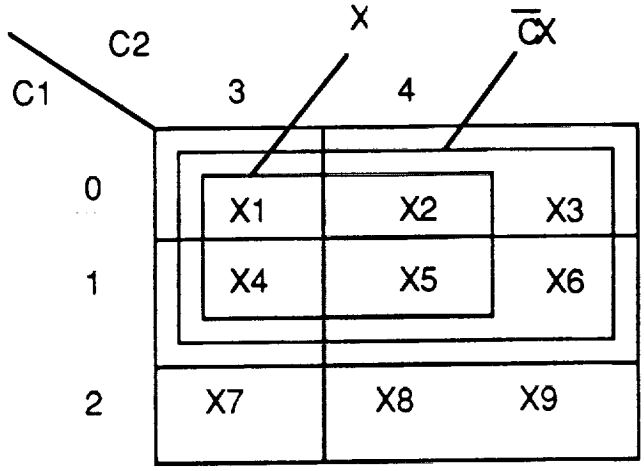


Figure (b) upper approximation $\overline{C}X$ of set X.

The set X determines the following rough set:

($\{x_1, x_2, x_4, x_5\}$, $\{x_1, x_2, x_4, x_6\}$, $\{x_1, x_3, x_4, x_5\}$, $\{x_1, x_3, x_4, x_6\}$).

For example, $x_1 \underline{\in} X$, hence $x_1 \overline{\in} X$, and $x_3 \underline{\notin} X$, but $x_3 \overline{\in} X$.

Now let us represent the decision of the expert from the example, corresponding to set X, by rules. Any such a rule is a conditional statement that specifies a decision under conditions. The smallest subsets of U which may be described by rules, using the set C of conditions, are the members of the classification generated by C. Therefore, we may represent set X by rules iff X is C-definable. If set X is C-undefinable we cannot represent it by a single set of rules. Instead, we may represent sets $\underline{C}X$ and $\overline{C}X$ by different sets of rules. In particular a rule derived from $\underline{C}X$ is **certain**, and a rule derived from $\overline{C}X$ is **possible**.

In the example, X is roughly C-definable in S. The **certain** rules, corresponding to set $\underline{C}X$ of positive examples and set -$\overline{C}X$ of negative examples, are

(Temperature, low) --> (Influenza, no)

(**Temperature**, **subfebrile**) ^ (**Headache**, **no**) --> (**Influenza**, **no**),

and the **possible rules**, corresponding to set $\overline{C}X$ of positive examples and set $-\overline{C}X$ of negative examples, are

(**Temperature**, **low**) --> (**Influenza**, **no**)

(**Headache**, **no**) --> (**Influenza**, **no**).

As can be seen from the above example, uncertainty is all too often present in the conditions and the decisions. The conditions and the decisions fail to partition the universe into well defined classes and some overlap is present. In real cases we do not have sharp boundaries between say **normal**, **subfebrile**, and **high**. The best we can hope is that **normal**, **subfebrile**, and **high**, "somewhat partition" the universe by not overlapping "too much." In the next chapter we will look at a method which would help us deal with such a setting, where attributes fail to have sharp boundaries.

CHAPTER 3

FUZZINESS AND FUZZY SETS

The way we humans actually store and manipulate concepts in the mind is a subject of some debate. However, we communicate conscious processes to other people verbally. This type of reasoning is done with words rather than numbers.This is why we find probability theory counter-intuitive, and in some cases difficult to understand. The many shades of meaning which give language its richness and colour contrast with the precise rigour of mathematical theory, logic and computer languages.

There is a difference between the meaning and usage of words. It is not the strict dictionary definition of a word which is important, but the way in which an expert uses a word. At times an expert may find it difficult to define a particular word, though usually he will be able to give an example of a use. We usually find technical terms relatively easy to define. However, commonly used words are less easy to define, either in abstract or even in context. For example let's consider the word "cold". What is the criterion for saying that the weather is cold? The answer depends on factors like temperature and the time of year. For instance a cold summer's day can be milder than a warm winter's day. It is relatively easy to quote examples of cold days and days which are not cold. There is a vagueness or fuzziness about a certain range of temperatures; they might constitute coldness, and they might not. In this chapter we take an in-depth look at this aspect of vagueness or fuzziness.

3.1 Fuzziness

Everyone uses fuzzy words in their everyday lives, and seldom question whether they or others understand their usage of those words. An individual may not be consistent in his own use of words, and there is even less chance that someone else has the same usage. Nevertheless, we all use words expressing belief when we are reasoning or arguing. For example, let us look at a quote from a doctor:

> "I wouldn't expect that disease in a young girl of 20. It's so rare as to be negligible. It isn't worth carrying out the test on a young person. If they're young I'd most likely not do the tests. If they are older I probably would do them."

Here the doctor is using vague rules. Some fuzzy words which he uses are young, older, negligible, so rare, most likely, probably etc. When pressed to define such words, experts often find it extremely difficult.

There is also a distinction between uncertainty and imprecision, which is not always reflected in the models used in computer systems. Uncertainty refers to something which is not known for sure, and imprecision refers to something whose value is not known accurately. Statements can be uncertain, imprecise or both. For example:

> "There will definitely be a rise in temperature: somewhere between 10 degrees and 25 degrees."

is imprecise but certain whereas:

> "I think you should leave it on. If so you should set it to 180 degrees."

is precise but uncertain.

3.1.1 IF .... THEN rules

Crisp mathematical rules can be easily defined. The basis for a rule is:

IF A then B or A --> B

This states that if A is true, then B is necessarily true too. It does not state that B implies A, and B can be true with A false. It is difficult to find a clear example of this concept except in the context of mathematics, for example:

if X = 2 then $X^2 = 4$

Note that $X^2 = 4$ does not mean that X is the value 2; X = -2 is another solution. The rule is exactly equivalent to:

Not B --> Not A

Unfortunately, in common usage "if" and "only if" are interchanged and used improperly all too often.

Statements based on logic are made more complex by the use of **AND** and **OR**. **AND** is easy to understand, but **OR** is ambiguous. If a child is told "You can have sweets or an ice cream", the child will usually understand that she is not allowed both. This is an exclusive **OR**. The statement "The leaves on the tree are green or yellow" implies that possibly some leaves are green and others are yellow. This is an inclusive **OR**: Yellow and green can occur together. The English language does not distinguish between these two meanings, and the interpretation may depend on the context. In formal logic and computer logic, the inclusive **OR** is more common. Further ambiguities arise when both terms **AND** and **OR** are used in the same statement. For example let us consider the rule:

"If the patient is over 40 and has high blood pressure or is female then I would refer them."

Does this statement mean:

"If the patient is over 40 and has high blood pressure or if the patient is over 40 and is female then I would refer them."

or does it mean:

"If the patient is over 40 and has high blood pressure or if the patient is female then I would refer them."

Only the person who made the statement can identify the correct interpretation. Note that the two interpretations give potentially different outcomes for a female patient under the age of 40. Again, in computer logic the meaning is unambiguous - the problem arises because of the way we use words.

3.1.2 Symptoms

Much judgement and reasoning using vague rules involve weighing up the strength of evidence in symptoms. For example:

"Meniere's disease causes spells of dizziness."

is a rule of the form:

If A then B

i.e. if you have Meniere's disease then you will have spells of dizziness. If we are told that a patient has spells of dizziness then it is more credible that he has Meniere's disease. However, dizziness can be caused by other illnesses or disorders. If dizziness is a common ailment for this type of patient then we do not have much evidence for Meniere's disease, but if it is rare except as a consequence of the disease, then our inference is stronger. The strength of our inference depends on how likely B is in itself. If B is very common, then we have little evidence for A; if B is very rare then A becomes much more credible. So B is true makes A more credible is our vague rule.

In practice, there is usually more than one symptom, or evidence, i.e. the rule is :

$A \dashrightarrow B_1, B_2, \ldots\ldots B_n$

For example:

"Meniere's disease causes spells of dizziness, tinnitus, and progressive hearing loss."

This form of reasoning is the one which is often represented by **Baye's rule**. The weights of evidence used in the doctor's diagnosis are not independent; it is a combination of symptoms which gives credibility to the solution.

3.1.3 Uncertainty in Data

The vagueness or uncertainty which is an intrinsic feature of judgement is not unique to rules. Data presented to an expert or expert system can also be uncertain. Some data are clear facts with a yes/no answer, for example:

The applicant is over the age of 18

but others may be fuzzy:

The patient may have suffered from indigestion

So expertise involves dealing with uncertain data, and uncertain inference rules using that data. Much of the skill in judgement lies in weighing up the relative merits of data, facts guesses and hypothesis, etc., and using a plausible line of reasoning with them. There are essentially two aspects to this uncertainty: belief and value. Belief is analogous to probability and measures the level of credibility whereas probability is a numerical measure. People generally use words to express belief. There are over 50 terms in the English language expressing belief, and the number can be increased by qualifiers such as very, extremely etc. However, if a subset of these terms could be agreed upon, together with an hierarchy expressing the relationships between them, then there is no reason why the expert should not be able to express his knowledge in simple English which is natural to him. For example the figure on the next page shows a simple hierarchy showing the relationships between terms such as possible, certain and definite. A term low down on the hierarchy is stronger than the one higher up. So 'certain' is stronger than 'probable', and proved implies 'definite'. The main problem with this is ascertaining whether the expert is consistent in his usage of words, and whether

the agreed relationships make sense to other people. The other element, that of value, is analogous to risk. Terms expressing value are those such as fatal, serious, dangerous, undesirable, etc. A possibility which is considered likely and serious may warrant immediate investigation, whereas one which is highly probable and undesirable may not. It will be necessary to draw up similar diagrams representing

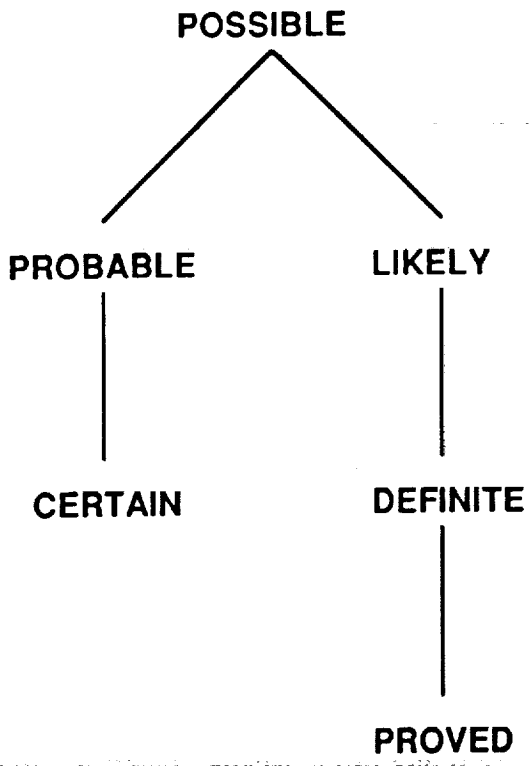


Figure (a) Hierarchy chart representing relationships between words.

relationships between words describing risk or value as well, if the uncertainty handling is to be written in words. If the expert can do this then it will usually be a valuable exercise. The problem, with using words is that there is such an abundance to choose from, but the advantage is that the language is easy for the expert to use. Risk is extremely important in reasoning processes. A low probability high risk

situation might warrant investigation before a high probability low risk one. It is the importance which matters. Reasoning seems to be multi-dimensional and probability theory on its own seldom provides an adequate framework. In other words objective probabilities do not embrace all facets of human judgement.

3.2 Fuzzy Sets

Even though numerical models for belief have many disadvantages, it cannot be denied that many famous expert systems do use them. Pure probability theory has been considered inadequate and some famous systems use certainty factors. Another important theory which is used in expert systems is the **fuzzy set theory**. Fuzzy set theory and fuzzy logic were formulated by Zadeh, and have since been applied to many problems where traditional crisp logic and mathematics are inappropriate because of the inherent uncertainty. In traditional logic a proposition is true or false; in fuzzy logic it has a degree of truth. For example, let us consider the question:

Is the object black? (or white)

In crisp logic the answer can be either yes (black) or no (white). In fuzzy logic an object would be given a degree of blackness, where 0 indicates 'definitely not' and 1 indicates 'definitely'. An off-white object could be measured by 0.2, say, and a grey object by 0.6. This would not mean that one was three times as black as the other, but would enable the members of a set to be ranked. Let U be the universe of discourse or domain:

$$U = U_1 + U_2 + \ldots\ldots\ldots + U_n$$

So U is the set of n objects U_1, U_2, ... U_n which we are considering. A fuzzy set F is described by its members and their degrees of membership to that set, for example:

$$F = M_1/U_1 + M_2/U_2 + \ldots\ldots\ldots + M_n/U_n$$

U_1, U_2, ... U_n are members with degrees of membership M_1, M_2, ... M_n, and + denotes union not addition. In other words this equation is a way of listing the various members together with their degrees of membership. Equivalently, F is given by:

$$F = \Sigma M_F(U_i)/U_i$$

where Σ denotes 'the set of'. We also define the fuzzy versions of union (inclusive OR), intersection(AND) and complement (NOT).

The grade of membership of U in the union $F \cup G$ (F OR G) is at least that of its membership in the individual sets F, G. We do not know any more than this, and so the grade of membership is given by the maximum of the two. So:

$$F \cup G = \Sigma M_F(U) V M_G(U)/U$$

where V denotes maximum. The grade of membership of U in $F \cap G$ (F AND G) can be no greater than the membership in each of F and G. So intersection is defined by:

$$F \cap G = \Sigma M_F(U) \wedge M_G(U)/U$$

where ^ denotes minimum. The value 1 denotes full membership and 0 no membership. The complement of F, F' is given by :

$$F' = \Sigma(1 - M_F(U))/U.$$

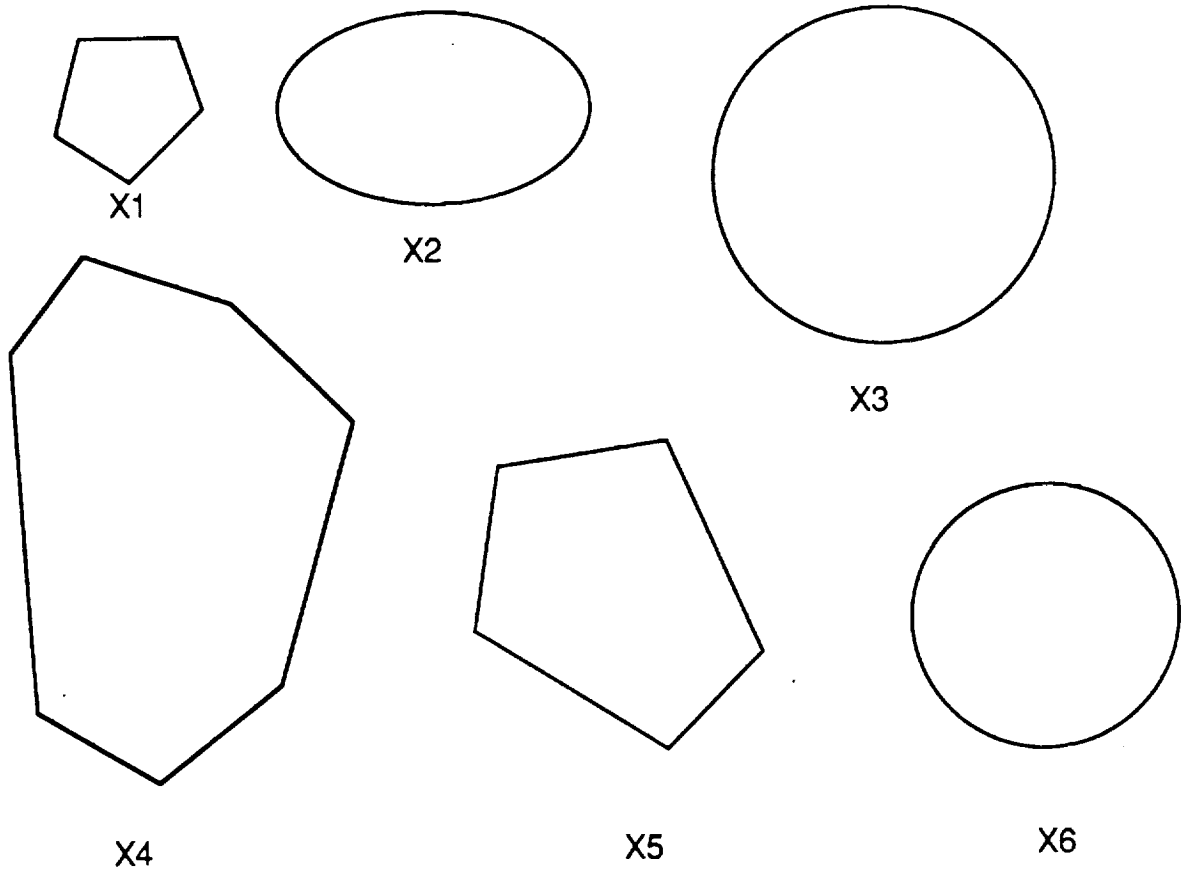


Figure (b) Set of six figures

Now let us look at an example by considering the objects in figure(b). Suppose L is the fuzzy set of large shapes, and R the fuzzy set of round shapes then L and R could be defined by:

$$L = 0.1/X_1 + 0.6/X_2 + 0.6/X_3 + 0.8/X_4 + 0.4/X_5 + 0.2/X_6$$

$$R = 0.1/X_1 + 0.7/X_2 + 1.0/X_3 + 0.5/X_4 + 0.1/X_5 + 1.0/X_6$$

$L \cup R$ is the set of objects which are large or round. X_1 is not really large and not particularly round, so its membership in $L \cup R$ is low. X_6 is not large but perfectly round so its membership in large or round is 1.

$$L \cup R = 0.1/X_1 + 0.7/X_2 + 1.0/X_3 + 0.8/X_4 + 0.4/X_5 + 1.0/X_6$$

$L \cap R$ is the set of objects which are large and round

$$L = 0.1/X_1 + 0.6/X_2 + 0.6/X_3 + 0.5/X_4 + 0.1/X_5 + 0.2/X_6$$

In this case X_5 and X_6 have low membership values for large and round ($L \cap R$) because membership in at least one of L and R is low. The strongest membership is for X_2 and X_3 both of which have fairly high membership in both L and in R together. L' is the set of not large (i.e. small) objects

$$L' = 0.9/X_1 + 0.4/X_2 + 0.4/X_3 + 0.2/X_4 + 0.6/X_5 + 0.8/X_6$$

So X_1 has a high membership in L' and X_4 has a low membership. The different membership values mean that it is not sensible to count the members in a fuzzy set. Instead we can define the power P of a set F by:

$$P(F) = \Sigma M_F(X)$$

So in our figure(b):

P(L) = 2.7

P(R) = 3.4

P(L∪R) = 4.0

In the next chapter we discuss the concepts of the present work, which is to extract fuzzy rules under uncertainty and to measure definibility using rough sets.

CHAPTER 4

EXTRACTING FUZZY RULES AND MEASURING DEFINIBILITY

The main purpose of the present work is to study a setting where a decision maker (expert) is faced with uncertain (i.e. fuzzy) symptoms and makes a fuzzy decision. Let's keep in mind that these decisions may be strongly or weakly based on the conditions. From the data we have, we will extract fuzzy rules, in fact we will extract two sets of rules i.e. **certain** and **possible** rules as well as a measure of how much we believe these rules. Finally we will define the decisions in terms of the symptoms. Before we go any further we will look at the properties, notations and operations of fuzzy sets which is required to understand this work.

4.1 Functions on pairs of fuzzy sets

We now look at some functions and properties of fuzzy sets. All the concepts explained here can be found in [2]. Let's recall from the previous chapter that a fuzzy subset A of U is defined by a characteristic function

$$\mu_A : U \dashrightarrow [0,1].$$

The notation

$$\sum_i \alpha_i/x_i \qquad (0 \le \alpha_i \le 1)$$

denotes a fuzzy subset whose characteristic function at x_i is α_i.

Moreover let us recall that if A and B are fuzzy subsets $A \cap B$, $A \cup B$, $-A$ are defined by $\text{Min}\{ \mu_A(x), \mu_B(x) \}$, $\text{Max}\{ \mu_A(x), \mu_B(x) \}$, and $1 - \mu_A(x)$ respectively. The implication

A -----> B is defined by -A$\cup$B and the characteristic function corresponding to -A$\cup$B is given by

$$\text{Max}\{\ 1 - A(x),\ B(x)\ \}.$$

Let us now go through an example and see how these work:

Let A = (.5, .7, .2, .4) and

Let B = (.4, .8, .9, .6)

then we have the following :

$$\text{Min}\{\ \mu_A(x),\ \mu_B(x)\ \} = (.4,\ .7,\ .2,\ .4)$$

$$\text{Max}\{\ \mu_A(x),\ \mu_B(x)\ \} = (.5,\ .8,\ .9,\ .6)$$

$$1 - \mu_A(x) = (.5,\ .3,\ .8,\ .6)$$

Now we look at two new functions on pairs of fuzzy sets.

$$I(A \subset B) = \inf_x \text{Max}\{1 - A(x),\ B(x)\ \}$$

$$J(A\#B) = \text{Max}_x\ \text{Min}\ \{A(x),\ B(x)\ \}.$$

where A and B denote fuzzy subsets of the same universe. The function I(A$\subset$B) measures the degree to which A is included in B and the function J(A#B) measures the degree to which A intersects B. If A and B are crisp sets it is evident that

$I(A \subset B) = 1$ if and only if $A \subset B$

otherwise it is 0.

Moreover in the case of crisp sets

$J(A\#B) = 1$ if and only if $A \cap B \neq \phi$

otherwise it is 0.

In addition to the above, lets also look at the following relation as shown in [2]

$$I(A \subset B) = 1 - J(A\#-B).$$

The right hand side of the above equation is

$$\inf_x \{1 - \text{Min}\{A(x),\ 1-B(x)\}\}$$

$$= \inf_x \text{Max}\{1-A(x),\ 1-(1-B(x))\}$$

$$= \inf_x \text{Max}\{1 - A(x),\ B(x)\}.$$

In the next section we go through an example and show step by step how fuzzy certain and possible rules can be extracted from raw data.

4.2 An Example

Let us consider the following table which is the kind of raw data we will be dealing with in this work, i.e. we will have a set of conditions and a set of decisions whose values are given using the fuzzy set theory.

Table 1

PATIENTS	SIZE	COLOR	DECISIONS
P1	3/L + .8/S	.2/R + .9/B	$.3/D_A + .6D_B$
P2	.4/L + .7/S	.4/R + .7/B	$.8/D_A + .5/D_B$
P3	7/L + .4/S	.6/R + .7/B	$.5/D_A + .9/D_B$
P4	.8/L + .5/S	.3/R + .8/B	$.7/D_A + .3/D_B$
P5	.2/L + .7/S	.2/R + .5/R	$.4/D_A + .2/D_B$
P6	.9/L + .2/S	.8/R + .2/B	$.7/D_A + .8/D_B$
P7	.3/L + .6/S	.7/R + .1/B	$.4/D_A + .5/D_B$

L = Large R = Red D_a = Disease A

S = Small B = Blue Db = Disease B

We will interpret the above table as a case where an expert is trying to determine the presence or absence of a disease by looking at the size and color of a tumor. The first column represents a number of patients i.e. P1, P2,....,P7. The symbols L and S stand for large and small respectively, and the symbols R and B stand for Red and Blue respectively. So we can interpret that patient P_1 has a tumor that is judged to be .3 large and .8 small. In this particular case, let us assume that a number of physicians are looking at the tumor, and that a certain number of them judge the tumor to be large, and others judge it to be small. So in our case the numbers .3 and .8 denote relative frequencies. However, it does not need to be so, i.e. these numbers could reflect some judgement and need not be generated as relative frequencies. The decision column shows a fuzzy diagnosis. So from our table, one's interpretation could be that patient P_1 is diagnosed to have disease D_A and the corresponding belief is .3. Also patient P_1 is diagnosed to have disease D_B and the corresponding belief is .6 strong.

Now what we want to do is to take these cases and unravel them into fuzzy rules as to when disease D_A or D_B is present. The first step is to take this raw data and convert them into fuzzy sets as follows:

$$D_A = .3/X_1 + .8/X_2 + .5/X_3 + .7/X_4 + .4/X_5 + .7/X_6 + .4/X_7$$

The fuzzy set for D_A is obtained by taking the union of the values of D_A of all the patients. Similarly fuzzy sets are created for large, small, red and blue as follows:

$$L = .3/X_1 + .4/X_2 + .7/X_3 + .8/X_4 + .2/X_5 + .9/X_6 + .3/X_7$$

$$S = .8/X_1 + .7/X_2 + .4/X_3 + .5/X_4 + .7/X_5 + .2/X_6 + .6/X_7$$

$$R = .2/X_1 + .4/X_2 + .6/X_3 + .3/X_4 + .2/X_5 + .8/X_6 + .7/X_7$$

$$B = .9/X_1 + .7/X_2 + .7/X_3 + .8/X_4 + .5/X_5 + .2/X_6 + .1/X_7$$

The next step would be to find the minimum degree to which possible combinations of symptoms imply disease D_A i.e. find the certain rules. This is done by computing

$I(L \subset D_A)$, $I(S \subset D_A)$, $I(R \subset D_A)$, $I(B \subset D_A)$, $I(L \cap R \subset D_A)$, $I(L \cap B \subset D_A)$, $I(S \cap R \subset D_A)$.

Similar computations would be carried out for D_B.

Carrying out the computations would yield the following results:

$I(L \subset D_A) = .5$	$I(L \subset D_B) = .3$	$I(S \subset D_A) = .3$
$I(S \subset D_B) = .3$	$I(R \subset D_A) = .4$	$I(R \subset D_B) = .5$
$I(B \subset D_A) = .3$	$I(B \subset D_B) = .3$	$I(L \cap R \subset D_A) = .5$
$I(L \cap R \subset D_B) = .6$	$I(L \cap B \subset D_A) = .5$	$I(L \cap B \subset D_B) = .3$
$I(S \cap R \subset D_A) = .4$	$I(S \cap R \subset D_B) = .5$	$I(S \cap B \subset D_A) = .3$
$I(S \cap B \subset D_B) = .5$		

Now all these yield **certain** rules. But we may not want to keep all the rules in order to avoid any partial implications. So we would set a threshold value, say for this example let us choose threshold value (α) to be .5. This would throw away any rule which evaluates below this threshold. Of course the lower the α is, the more partial implications are taken into account. The choice of α is very much problem dependent. So after applying the threshold value the **certain rules** we are left with are as follows: If the tumor is large then D_A is present is 0.5

If the tumor is large and red then D_A is present is 0.5

If the tumor is large and blue then D_A is present is 0.5

If the tumor is red then D_B is present is 0.5

If the tumor is large and red then D_B is present is 0.6

If the tumor is small and red then D_B is present is 0.5

If the tumor is small and blue then D_B is present is 0.5

Next we find the possible rules by using the second function which is J(X#Y). Again we choose a threshold value and discard any rules which falls below this threshold value. These values measure the degree to which X intersects Y, and the rules generated by these are the possible rules. So carrying out the computations we get :

$J(S \# D_A) = .7$	$J(S \# D_B) = .6$	$J(R \# D_B) = .7$
$J(R \# D_B) = .8$	$J(B \# D_A) = .7$	$J(S \# D_B) = .7$
$J(L \cap R \# D_A) = .7$	$J(L \cap R \# D_B) = .8$	$J(L \cap B \# D_A) = .7$
$J(L \cap B \# D_B) = .7$	$J(S \cap R \# D_A) = .4$	$J(S \cap R \# D_B) = .5$
$J(S \cap B \# D_A) = .7$	$J(S \cap B \# D_B) = .7$	

For the possible rules let us set the threshold value (α) to .6. This would yield the following possible rules.

If the tumor is large then D_A is possible 0.7

If the tumor is small then D_A is possible 0.7

If the tumor is red then D_A is possible 0.7

If the tumor is blue then D_A is possible 0.7

If the tumor is large and red then D_A is possible 0.7

If the tumor is large and blue then D_A is possible 0.7

If the tumor is small and blue then D_A is possible 0.7

If the tumor is large then D_B is possible 0.8

If the tumor is small then D_B is possible 0.6

If the tumor is red then D_B is possible 0.8

If the tumor is blue then D_B is possible 0.7

If the tumor is large and red then D_B is possible 0.8

If the tumor is large and blue then D_B is possible 0.7

If the tumor is small and blue then D_B is possible 0.6

Now we are ready to extract the certain and possible rules. In the next section we look at how this is done.

4.3 Extracting Rules

The method used for extracting rules differ for the certain and possible rules. We will look at each case individually. First we look at how to extract certain rules. To extract certain rules:

1) All rules with unique degrees of belief are kept.
2) In case two or more rules have the same degree of belief then the one with the smaller number of attributes are kept.

Applying these rules to the above stated certain rules we get the following extracted certain rules.

If the tumor is large then D_A is present is 0.5

If the tumor is red then D_B is present is 0.5

If the tumor is large and red then D_B is present is 0.6

If the tumor is small and blue then D_B is present is 0.5

Now we see how to extract possible rules. The steps are as follows:

1) All rules with unique degrees of belief are kept.
2) In case two or more rules have the same degree of belief then the one with the larger number of attributes are kept.

Applying these rules to the possible rules shown above we get the following extracted possible rules:

If the tumor is large and red then D_A is possible 0.7

If the tumor is large and blue then D_A is possible 0.7

If the tumor is small and blue then D_A is possible 0.7

If the tumor is large and red then D_B is possible 0.8

If the tumor is large and blue then D_B is possible 0.7

If the tumor is small and blue then D_B is possible 0.6

4.4 Measuring Definibility

Now we turn our attention to defining the fuzzy terms involved in the diagnosis as a function of the terms used in the symptoms. How well we are able to do this , is a function of how much the decision follows the conditions. The concepts explained here are from [2]. Let $\{B_i\}$ be a finite family of fuzzy sets which does not necessarily form a partition of the universal set. Let A be a fuzzy set. Then we can define the lower approximation of A through $\{B_i\}$ as

$$\underline{R}(A) = \bigcup_i I(B_i \subset A) B_i.$$

In cases where $I(B_i \subset A)$ is less than some threshold α it is advantageous to throw away all the sets B_i. In this case we have

$$\underline{R}(A)_\alpha = \bigcup I(B_i \subset A)_\alpha B_i.$$

Similarly we show an upper approximation of A through {Bi}

$$\overline{R}(A) = \bigcup_i (B_i \# A) B_i$$

and

$$\overline{R}(A)_\alpha = \bigcup (B_i \# A)_\alpha B_i$$

Returning to our initial example and applying these concepts: if we choose α to be .5 and $B_1 = L$; $B_2 = L \cap R$; $B_3 = L \cap B$; $B_4 = S \cap B$ then we have

$$\underline{R}(D_A)_{.5} = .5\,L \cup .5(S \cap B)$$

Thus we can use a combination of Large and Small and Blue can to describe the set of patients that are certainly sick through the symptoms L,S,R,B. Similarly, if we pick α to be .6 then we get

$$\overline{R}(D_A)_{.6} = .7L \cup .7S \cup .7R \cup .7B$$

SUMMARY

In this work we looked at a method which allows us to acquire knowledge from data which contains uncertainty or missing and/or erroneous data. The knowledge is acquired by looking at examples. These examples are more or less like an information system, that is it contains a set of conditions and a set of decisions. These conditions and situations contain attributes whose values are subject to judgement, i.e. the attributes are not crisp, but are fuzzy sets. From this raw data, which contains uncertainty, we looked at how we could extract rules as well as measures of how much we trust these rules. In fact we generated two kinds of rules. The two sets of rules we generated were as follows, those generated by certainty and those generated by possibility. Finally we looked at how we could define the fuzzy terms involved in the decisions as a function of the terms used in the conditions. A program which incorporates all this is given at the Appendix section.

REFERENCES

[1] Cohen, P.R. and Grinberg, M. R., A framework for heuristic reasoning about uncertainty, *Proc. 8th IJ CAI, Karlsruhe, W. Germany*, pp. 355 - 357 (1983).

[2] de Korvin, Andre', Bourgeois, B., Kleyle, R. *Extracting Fuzzy rules under Uncertainty and measuring definibility using rough sets*, Technical Report, University of Houston - Downtown, Houston, Tx.

[3] Ehrenberg, ASC (1982) *A Primer in Data Reduction* John Wiley, Chichester.

[4] Grzymala-Busse, J. W. "Knowledge acquisition under uncertainty - a rough set approach," Journal of Intelligent and Robotic Systems 1, pp. 3 -16 (1988).

[5] Grzymala-Busse, J. W., On the reduction of knowledge representation systems, *Proc. 6th Internat. Workshop on Expert Systems and their Applications, Avignon , France.* Vol. 1, pp. 587- 599 (1986).

[6] Johnson-Laird, PN and Wason, Pc (Eds) (1977) *Thinking : Readings in Cognitive Science* Cambridge University Press.

[7] McCarthy, J., Circumscription - a form of non-monotonic reasoning, *AI* 13, 27-39 (1980).

[8] McDermott, D. and Doyle, J., Non-monotonic logic I *AI* 13, 41-72 (1980).

[9] Mamdani, A., Efstathiou, J., and Pang, D., Inference under uncertainty, *Expert Systems 85 . Proc. Fifth Tech. Conf. British Computer Soc., Specialist Group on Expert Systems*, pp. 181-194(1985).

[10] Shafer, G., *Mathematical theory of Evidence*, Princeton University Press (1976).

APPENDIX

A PROGRAM WHICH EXTRACTS FUZZY RULES

```
Program Extract_Fuzzy_Rules(Input,Output);

  {*************************************************************************}
  {*************************************************************************}
  {******                                                           ******}
  {******  PROGRAM FOR PARTIAL FULFILLMENT OF THE REQUIREMENTS      ******}
  {******                      FOR CS 4395                          ******}
  {******                                                           ******}
  {******            Programmers :  Donald Culas                    ******}
  {******                           Jeff Worm                       ******}
  {******                                                           ******}
  {******       This program simulate the ideas set forth in        ******}
  {******  Dr. Andre de Korvin's paper, "Extracting Fuzzy Rules     ******}
  {******  under uncertainty and Measuring Definibility using       ******}
  {******  Rough Sets".  The program is designed to combine the     ******}
  {******  methods of rough sets and fuzzy sets to measure          ******}
  {******  uncertainty.  Fuzzy rules are extracted to provide       ******}
  {******  the user a foundation from which they may formulate      ******}
  {******  a decision.                                              ******}
  {******                                                           ******}
  {*************************************************************************}
  {*************************************************************************}

const
  max_cases = 100;

type
  str  = string [10];
  fuzzy_array = array[1..max_cases] of real;

struct  =  record
  att1       : str;
  att2       : str;
  value_att1 : real;
  value_att2 : real;
end;

cond_struct = record
   consider  : str;
   firattr   : str;
   secattr   : str;
   firattr_1 : str;
   firattr_2 : str;
   secattr_1 : str;
   secattr_2 : str;
end;


dec_struct = record
  consider : str;
  firattr  : str;
  secattr  : str;
end;

value_struct = record
   kind_1      : str;
   kind_2      : str;
   dec_kind    : str;
   value       : real;
   attr        : integer;
   tag         : integer;
end;

case_struct  =  record
     condition_1       :  struct;
```

```
    decision          :  struct;
end;

info = array[1..max_cases] of case_struct;
value_array = array [1..9] of value_struct;


var
   cases  : info;
   thresh : real;
   m_ch,sit_ch,d_ch,val_ch,see_ch : char;
   no_of_cases,count : integer;
   read_data : boolean;
   condition : cond_struct;
   decision  : dec_struct;
   cond1_att1_arr,cond1_att2_arr,cond2_att1_arr,cond2_att2_arr : fuzzy_ar
   dec_att1_arr,dec_att2_arr,inter_array : fuzzy_array;
   dec1_sub,dec2_sub,dec1_inter,dec2_inter : value_array;


{*************************************************************************}
{************************  PROCEDURE INITIALIZE  ************************}
{*************                                                ***********}
{*************  This procedure initializes the strings.  ***************}
{*************************************************************************}
procedure initialize(var con : cond_struct;
                     var dec : dec_struct);

var
   blank : string [10];
begin                                          { INITIALIZE }
  blank := '          ';
  con.consider := blank;
  con.firattr  := blank;
  con.secattr  := blank;
  con.firattr_1  := blank;
  con.firattr_2  := blank;
  con.secattr_1  := blank;
  con.secattr_2  := blank;
  dec.consider  := blank;
  dec.firattr  := blank;
  dec.secattr  := blank;
end;                                           { INITIALIZE }


{*************************************************************************}
{******************  PROCEDURE READ SITUATION  ************************}
{****                                                                 ****}
{****  This procedure reads in the conditional and decisional       ****}
{****  attributes that the user enters.                             ****}
{*************************************************************************}
procedure read_situation(var con : cond_struct;
                         var dec : dec_struct);
                                               { READ SITUATION }
begin
  writeln;
  writeln;
  writeln;
  writeln('Please enter the attribute under consideration ');
  write(' (eg.tumor, weather, etc... ) : ' );
  readln(con.consider);
  writeln;
  write('Please enter the decision attribute (eg. disease, factory, etc..
');
  readln(dec.consider);
  writeln;
```

```
  writeln('Please enter the two attributes of ',con.consider,' ', 'we will be
oking at');
  write('First attribute of ',con.consider, ' : ');
  readln(con.firattr);
  write('Second attribute of ',con.consider,' : ');
  readln(con.secattr);
  writeln;
  writeln;
  writeln('Please enter two attributes for ',con.firattr);
  write('First attribute for ',con.firattr,' : ');
  readln(con.firattr_1);
  write('Second attribute for ',con.firattr,' : ');
  readln(con.firattr_2);
  writeln;
  writeln;
  writeln('Please enter two attributes for ',con.secattr);
  write('First attribute for ',con.secattr,' : ');
  readln(con.secattr_1);
  write('Second attribute for ',con.secattr,' : ');
  readln(con.secattr_2);
  writeln;
  writeln;
  writeln('Please enter two attributes for ',dec.consider);
  write('First attribute for ',dec.consider,' : ');
  readln(dec.firattr);
  write('Second attribute for ',dec.consider,' : ');
  readln(dec.secattr);
end;                                          { READ SITUATION }



{*****************************************************************}
{****************  PROCEDURE SET FUZZY VALUE  ********************}
{****                                                          ****}
{****  This procedure reads in real data (i.e. 20, 58, 265)    ****}
{****  and converts it into fuzzy values - values between 0    ****}
{****  and 1 (i.e. 0.7, 0.3, 0.28).                            ****}
{*****************************************************************}
procedure set_fuzzy_value(var val,max,min,first_val,sec_val : real);
   begin                                      { SET FUZZY VALUE }
     if (val > max ) then
        begin
           first_val := 1;
           sec_val := 0;
        end
     else
      if ( val < min ) then
        begin
           sec_val := 1;
           first_val := 0;
        end
     else
        begin
           first_val := (val - min)/(max-min);
           sec_val := (max - val)/(max-min);
        end;
   end;                                       { SET FUZZY VALUE }


{*****************************************************************}
{************************  READ REAL  ****************************}
{****                                                          ****}
{****  This procedure reads in the real (not fuzzy) values and ****}
{****  requires the user to establish high and low values for  ****}
{****  the conditional and decisional attributes.              ****}
```

```
{*********************************************************************}
procedure read_real ( con : cond_struct;
                      dec : dec_struct;
                      var case_arr : info;
                      var n : integer);

var
    firattr_1,firattr_2,secattr_1,secattr_2,dec_1,dec_2,value : real;
    i  : integer;
    ch : char;
                                                  { READ REAL }
begin
   ch := 'y';
    i := 1;
   writeln;
   write('Please enter a number you associate with definitely ',con.firattr_1
: ');
   readln(firattr_1);
   write('Please enter a number you associate with definitely ',con.firattr_2
: ');
   readln(firattr_2);
   writeln;
   write('Please enter a number you associate with definitely ',con.secattr_1
: ');
   readln(secattr_1);
   write('Please enter a number you associate with definitely ',con.secattr_2
: ');
   readln(secattr_2);
   writeln;
   write('Please enter a number you associate with definitely ',dec.firattr_1
');
   readln(dec_1);
   write('Please enter a number you associate with definitely ',dec.secattr_1
');
   readln(dec_2);
   WHILE ( (ch = 'y') or (ch = 'Y') ) do
      begin
         case_arr[i].condition_1.att1 := con.firattr_1;
         case_arr[i].condition_1.att2 := con.firattr_2;
         case_arr[i].condition_2.att1 := con.secattr_1;
         case_arr[i].condition_2.att2 := con.secattr_2;
         case_arr[i].decision.att1 :=    dec.firattr;
         case_arr[i].decision.att2 :=    dec.secattr;
         write('Please enter a value for ',con.firattr, ' : ');
         readln(value);
         set_fuzzy_value(value,firattr_1,firattr_2,case_arr[i].condition_1.va
_att1,case_arr[i].condition_1.value_att2);
         write('Please enter a value for ',con.secattr, ' : ');
         readln(value);
         set_fuzzy_value(value,secattr_1,secattr_2,case_arr[i].condition_2.va
_att1,case_arr[i].condition_2.value_att2);
         write('Please enter a value for ',dec.consider, ' : ');
         readln(value);
         set_fuzzy_value(value,dec_1,dec_2,case_arr[i].decision.value_att1,ca
arr[i].decision.value_att2);
         writeln;
         i := i + 1;
   REPEAT                { Makes user enter 'Y' or 'S' }
      writeln;
      write('Please enter [y] to input more data or [s] to stop : ');
      readln(ch);
   UNTIL ( (ch = 's') or (ch = 'S') or (ch = 'y') or (ch = 'Y') );
      end;
      ch := ' ';
       n := i-1;
                                                  { READ REAL }
end;
```

```
{*************************************************************************}
{***************  PROCEDURE CREATE CONDITION_1 FUZZY SETS  ***************}
{****                                                                  ****}
{****  This procedure creates the fuzzy sets for the first condition.  ****}
{*************************************************************************}
procedure create_cond1_fuzzy_sets( con : info;
                                   n   : integer;
                                   var new_arr_1 : fuzzy_array;
                                   var new_arr_2 : fuzzy_array );

var
  i : integer;
                                                  { create_cond1_fuzzy_sets }
begin
  for i := 1 to n do
    begin
      new_arr_1[i] := con[i].condition_1.value_att1;
      new_arr_2[i] := con[i].condition_1.value_att2;
    end;
                                                  { create_cond1_fuzzy_sets }
end;


{*************************************************************************}
{******************  CREATE CONDITION_2 FUZZY SETS  **********************}
{****                                                                  ****}
{**** This procedure creates the fuzzy sets for the second condition.  ****}
{*************************************************************************}
procedure create_cond2_fuzzy_sets( con : info;
                                   n   : integer;
                                   var  new_arr_1 : fuzzy_array;
                                   var  new_arr_2 : fuzzy_array);

var
  i  : integer;
                                      { CREATE COND_2 FUZZY SETS }
begin
  for i := 1 to n do
    begin
      new_arr_1[i] := con[i].condition_2.value_att1;
      new_arr_2[i] := con[i].condition_2.value_att2;
    end;
                                      { CREATE COND_2 FUZZY SETS }
end;


{*************************************************************************}
{**************  PROCEDURE CREATE DECISION FUZZY SETS  *******************}
{***                                                                   ****}
{*** This procedure creates the fuzzy sets of the decisional attributes ***}
{*************************************************************************}
procedure create_decision_fuzzy_sets(con  : info;
                                     n    : integer;
                                     var  new_arr_1 : fuzzy_array;
                                     var  new_arr_2 : fuzzy_array);

var
  i  : integer;
                                      { CREATE DECISION FUZZY SETS }
begin
  for i := 1 to n do
    begin
       new_arr_1[i]  := con[i].decision.value_att1;
       new_arr_2[i]  := con[i].decision.value_att2;
    end;
                                      { CREATE DECISION FUZZY SETS }
end;


{*************************************************************************}
{************************  PROCEDURE INIT  *******************************}
{****                                                                  ****}
{****        This procedure initializes the strings.                   ****}
{*************************************************************************}
```

```
procedure init(var arr : value_array);
var
   i  : integer;
   blank : string [10];
                                        { PROCEDURE INIT }
begin
   blank :=  '          ';
   for i := 1 to 9 do
      begin
         arr[i].kind_1 := blank;
         arr[i].kind_2 := blank;
         arr[i].tag := -1;
      end;
                                        { PROCEDURE INIT }
end;


{*************************************************************************}
{******************  PROCEDURE SET CONDITIONS  ***************************}
{****                                                                  ****}
{****      This procedure assigns the conditional attributes           ****}
{*************************************************************************}
procedure set_cond ( con  :  cond_struct;
                     var arr1 : value_array);
                                        { SET CONDITION }
begin
   arr1[1].kind_1 := con.firattr_1;
   arr1[1].attr := 2;
   arr1[2].kind_1 := con.firattr_2;
   arr1[2].attr := 2;
   arr1[3].kind_1 := con.secattr_1;
   arr1[3].attr  := 2;
   arr1[4].kind_1 := con.secattr_2;
   arr1[4].attr := 2;
   arr1[5].kind_1 := con.firattr_1;
   arr1[5].kind_2 := con.secattr_1;
   arr1[5].attr := 3;
   arr1[6].kind_1 := con.firattr_1;
   arr1[6].kind_2 := con.secattr_2;
   arr1[6].attr := 3;
   arr1[7].kind_1 := con.firattr_2;
   arr1[7].kind_2 := con.secattr_1;
   arr1[7].attr := 3;
   arr1[8].kind_1 := con.firattr_2;
   arr1[8].kind_2 := con.secattr_2;
   arr1[8].attr := 3;
                                        { SET CONDITION }
end;


{*************************************************************************}
{*********************  PROCEDURE SET DECISION_1  ************************}
{****                                                                  ****}
{****    This procedure assigns the first decisional attribute         ****}
{*************************************************************************}
procedure set_dec1(dec : dec_struct;
                   var   arr : value_array);

var
   i  : integer;
                                        { SET DECISION_1 }
begin
   for i := 1 to 8 do
     arr[i].dec_kind := dec.firattr;
                                        { SET DECISION_1 }
end;


{*************************************************************************}
{*********************  PROCEDURE SET DECISION_2  ************************}
```

```
{****                                                                    ****}
{****     This procedure assigns the second decisional attribute         ****}
{************************************************************************}
procedure set_dec2(dec : dec_struct;
                   var   arr : value_array);

var
   i  : integer;
begin                                          { SET DECISION_2 }
   for i := 1 to 8 do
     arr[i].dec_kind := dec.secattr;
end;                                           { SET DECISION_2 }



{************************************************************************}
{*************************  FUNCTION SUB  *******************************}
{****                                                                    ****}
{****  This function evaluates the values of I (A c B).  This value      ****}
{****  is the degree to which A is included in B and is used to          ****}
{****  generate the certain rules.                                       ****}
{************************************************************************}
function sub ( j : integer;
               arr1, arr2 : fuzzy_array) : real;
var
  temp : fuzzy_array;
  min  : real;
  i    : integer;
begin                                          { FUNCTION SUB }
   for i := 1 to j do
      begin
         temp[i] := 1 - arr1[i];
         if (arr2[i] > temp[i] ) then
             temp[i] := arr2[i];
      end;
   min := temp[1];
   for i := 2 to j do
      begin
         if ( temp[i] < min ) then
              min := temp[i];
      end;
   sub := min;
end;                                           { FUNCTION SUB }



{************************************************************************}
{*************************  FUNCTION NUM  *******************************}
{****                                                                    ****}
{****  This function evaluates the values of J(A # B). This value       ****}
{****  is the degree to which A intersects B and is used to              ****}
{****  generate the possible rules.                                      ****}
{************************************************************************}
function num ( j : integer;
               arr1, arr2 : fuzzy_array) : real;
var
  temp : fuzzy_array;
  i    : integer;
  max  : real;
begin                                          { FUNCTION NUM }
   for i := 1 to j do
      begin
         if ( arr1[i] <= arr2[i] ) then
              temp[i] := arr1[i]
         else
```

```
            temp[i] := arr2[i]
      end;
   max := temp[1];
   for i := 2 to j do
      begin
         if ( temp[i] > max ) then
              max := temp[i];
      end;
   num := max;
end;                                                    { FUNCTION NUM }


{*************************************************************************}
{*****************  PROCEDURE INTERMEDIATE  ******************************}
{****                                                                    ****}
{****  This procedure sets the values in the intermediate array          ****}
{****  for the values of I(A c B) and J(A # B).                          ****}
{*************************************************************************}
procedure inter ( j : integer;
                  arr1,  arr2 : fuzzy_array;
                  var temp_array : fuzzy_array);

var
   i : integer;
begin                                                   { INTER }
   for i := 1 to j do
      begin
         if ( arr1[i] <= arr2[i] ) then
           temp_array[i] := arr1[i]
         else
           temp_array[i] := arr2[i]
      end;
end;                                                    { INTER }


{*************************************************************************}
{**********************  PROCEDURE VALUE SUB  ****************************}
{****                                                                    ****}
{****  This procedure assigns the values to the corresponding terms      ****}
{****  of I(A c B).  It calls on the Function SUB.                       ****}
{*************************************************************************}
procedure value_sub( n : integer;
                     co1att1,co1att2,co2att1 : fuzzy_array;
                     co2att2,decatt : fuzzy_array;
                     var  new_arr : value_array);

var
   temp_arr : fuzzy_array;
                                                        { VALUE SUB }
begin
   new_arr[1].value := sub(n,co1att1,decatt);
   new_arr[2].value := sub(n,co1att2,decatt);
   new_arr[3].value := sub(n,co2att1,decatt);
   new_arr[4].value := sub(n,co2att2,decatt);
   inter(n,co1att1,co2att1,temp_arr);
   new_arr[5].value := sub(n,temp_arr,decatt);
   inter(n,co1att1,co2att2,temp_arr);
   new_arr[6].value := sub(n,temp_arr,decatt);
   inter(n,co1att2,co2att1,temp_arr);
   new_arr[7].value := sub(n,temp_arr,decatt);
   inter(n,co1att2,co2att2,temp_arr);
   new_arr[8].value := sub(n,temp_arr,decatt);
                                                        { VALUE SUB }
end;
```

```
{*************************************************************************}
{*********************  PROCEDURE INITIALIZE TAG  ************************}
{****                                                                  ****}
{****            This procedure initializes the PRINT tag              ****}
{*************************************************************************}
procedure init_tag( var arr1 : value_array);
   var
      i : integer;
   begin                                          { INITIALIZE TAG }
      for i := 1 to 8 do
       arr1[i].tag := -1;
   end;                                           { INITIALIZE TAG }


{*************************************************************************}
{*********************  PROCEDURE VALUE INTERMEDIATE  ********************}
{****                                                                  ****}
{****  This procedure assigns the values to the corresponding terms    ****}
{****  of J(A # B).  It calls on the Function NUM.                     ****}
{*************************************************************************}
procedure value_inter(n : integer;
                      colatt1,colatt2,co2att1 : fuzzy_array;
                      co2att2,decatt : fuzzy_array;
                      var new_arr : value_array);

var
  temp_array  : fuzzy_array;
                                                  { VALUE INTER }
begin
   new_arr[1].value := num(n,colatt1,decatt);
   new_arr[2].value := num(n,colatt2,decatt);
   new_arr[3].value := num(n,co2att1,decatt);
   new_arr[4].value := num(n,co2att2,decatt);
   inter(n,colatt1,co2att1,temp_array);
   new_arr[5].value := num(n,temp_array,decatt);
   inter(n,colatt1,co2att2,temp_array);
   new_arr[6].value := num(n,temp_array,decatt);
   inter(n,colatt2,co2att1,temp_array);
   new_arr[7].value := num(n,temp_array,decatt);
   inter(n,colatt2,co2att2,temp_array);
   new_arr[8].value := num(n,temp_array,decatt);
end;                                              { VALUE INTER }


          {*************************************************}
          {*************************************************}
          {*****                                       *****}
          {*****  The following four procedures are    *****}
          {*****  simply procedures to produce the     *****}
          {*****  headings for the output.             *****}
          {*****                                       *****}
          {*************************************************}
          {*************************************************}

procedure printall_head;
   begin
      writeln;
      writeln;
      writeln('   These are all the rules  ');
      writeln(' --------------------------');
      writeln(' --------------------------');
   end;

procedure thresh_head ( tval : real);
   begin
      writeln;
```

```
      writeln;
      writeln('  These are the rules after applying the threshold value of ',tv
1:2:1);
      writeln(' ------------ ----------------------------------------------------
-------');
      writeln(' ----------------------------------------------------------------
-------');
   end;

procedure certain_head;
   begin
      writeln;
      writeln(' The following are the certain rules ');
      writeln('-------------------------------------');
      writeln;
   end;

procedure possible_head;
   begin
      writeln(' The following are the possible rules ');
      writeln('-------------------------------------');
      writeln;
   end;


{*************************************************************************}
{**********************  PROCEDURE KIND RULES  ***************************
{****                                                                  ****}
{****  This procedure asks the user what kind or rules they will use ****}
{*************************************************************************}
procedure kind_rules(var ch : char);
   begin                                          { KIND RULES }
      writeln;
      writeln;
      ch := ' ';
    REPEAT
      writeln( 'What kind of rules would you like to see ?');
      writeln(' ');
      writeln(' Please enter [a] for all the rules : ');
      writeln('                or ');
      write(' Please enter [e] for the extracted rules: ');
      readln(ch);
    UNTIL ( (ch = 'a') or (ch = 'A') or (ch = 'e')  or (ch = 'E'));
      writeln;
      writeln;
      writeln;
   end;                                           { KIND RULES }


{*************************************************************************}
{*********************  PROCEDURE CHECK THRESHOLD  ***********************}
{****                                                                  ****}
{****  This procedure asks the user if they want to set a threshold  ****}
{****  value for the certain or possible rules.                      ****}
{*************************************************************************}
procedure check_thresh( var ch : char;
                        c_or_p : char);
begin                                             { CHECK THRESHOLD }
   ch := ' ';
   writeln;
   writeln;
   write('Do you want to see only the ');
     if (c_or_p = 'c') then
   write('certain ')
```

```
      else
    write('possible ');
    writeln ('rules above ');
    writeln('        a threshold value? ');
    writeln;
   REPEAT
    writeln;
    writeln;
    write('Please enter [y] to see only the ');
      if (c_or_p = 'c') then
    write('certain ')
      else
    write('possible ');
    writeln('rules which ');
    writeln('are above a threshold value  ');
    writeln('                    or   ');
    write('Enter [n] to see all the ');
      if (c_or_p = 'c') then
       write('certain ')
     else
    write('possible ');
    write(' rules : ');
    readln(ch);
    UNTIL (( ch = 'y') or ( ch = 'Y') or (ch = 'n') or (ch = 'N'));
end;                                              { CHECK THRESHOLD }


{************************************************************************}
{******************  PROCEDURE GET THRESHOLD VALUE  *********************}
{****                                                                  ****}
{****  This procedure reads the threshold value if one is entered.   ****}
{************************************************************************}
procedure get_tval( var num : real;
                    var t_kind : char);
begin                                             { GET T_VAL }
    write('Please enter the threshold value for the ');
      if (t_kind = 'c') then
    write( 'certain ')
      else
    write('possible ');
    write('rules : ');
    readln(num);
end;                                              { GET T_VAL }


          {************************************************}
          {************************************************}
          {******                                    ******}
          {******  The following two procedures are  ******}
          {******  headings for extracted and/or     ******}
          {******  certain rules.                    ******}
          {******                                    ******}
          {************************************************}
          {************************************************}

procedure extract_head;
   begin
     writeln;
     writeln;
     writeln(' These are the extracted rules ');
     writeln('-------------------------------');
     writeln('-------------------------------');
     writeln;
   end;
```

```
procedure certain_extract_head;
   begin
      writeln(' ');
      writeln(' ');
      writeln(' These are the extracted certain rules ');
      writeln('-------------------------------------');
      writeln;
   end;


{*************************************************************************}
{*****************   PROCEDURE PRINT ENGLISH  ************************}
{****                                                                  ****}
{****  This procedure produces the generated rules in english.       ****}
{*************************************************************************}
procedure print_english ( arr1 : value_array;
                          c_or_p : char;
                          alpha  : real;
                          kind   : char;
                          arr_size : integer);

var
   i     : integer;
   print : boolean;
   blank : string [10];
                                                  { PRINT ENGLISH }
begin
   blank := '          ';
   for i := 1 to arr_size do
      begin
         print := false;
         if ( ( kind = 'a') or ( kind = 'A') ) then
            begin
               if (arr1[i].value >= alpha) then
               print := true
            end
      else
         if ( (kind = 'e') or (kind = 'E') ) then
           if ( (arr1[i].value >= alpha ) and ( arr1[i].tag = 1) ) then
                    print := true;
             if print then
                begin
                   write('If the ',condition.consider,' is ',arr1[i].kind_1);
                   if ( arr1[i].kind_2 <> blank ) then
                       write (' and ',arr1[i].kind_2);
                       write(' then ',decision.consider,' ', arr1[i].dec_kind);
                   if ( ( c_or_p = 'c') or ( c_or_p = 'C') ) then
                       write( ' is present ')
                 else
                       write ( ' is possible ');
                       writeln(arr1[i].value:2:1);
                       writeln;
                end;
      end;
    writeln;
    writeln;
    writeln;
    writeln;
                                                  { PRINT ENGLISH }
end;


{*************************************************************************}
{*****************  PROCEDURE POSSIBLE EXTRACT HEADER  *****************}
{****                                                                  ****}
{****  This procedure generates the heading for extracted rules.     ****}
```

```
{*************************************************************************}
procedure possible_extract_head;
                                                  { POSSIBLE EXTRACT HEAD }
   begin
      writeln(' ');
      writeln(' ');
      writeln(' These are the extracted possible rules ');
      writeln('---------------------------------------');
      writeln;
                                                  { POSSIBLE EXTRACT RULES }
   end;


{*************************************************************************}
{*********************  PROCEDURE EXTRACT RULES  *************************}
{****                                                                 ****}
{****  This procedure extracts the rules.  First, all rules with      ****}
{****  unique I(AcB) and J(A#B) values are kept.  Secondly, if more   ****}
{****  than one rule has identical I values, the "smaller" in terms   ****}
{****  of attributes is kept. Conversely, the "larger" rules are      ****}
{****  kept when dealing with identical J values.                     ****}
{*************************************************************************}
procedure extract_rules(var arr1   : value_array;
                            c_or_p : char;
                            thresh_value : real);

var
   n,i,j : integer;
                                                  { EXTRACT RULES }
begin
   n := 9;
   for i := 1 to n-1 do
      begin
         if ( arr1[i].value >= thresh_value ) then
           begin
              for j := i + 1 to n do
                 begin
                   if ( arr1[i].attr <> arr1[j].attr ) then
                     begin
                        if ( c_or_p = 'c' ) then
                           begin
                              if ((( arr1[i].kind_1 = arr1[j].kind_1 ) or
                                   ( arr1[i].kind_1 = arr1[j].kind_2 )) and
                                   ( arr1[i].value = arr1[j].value) ) then
                               begin
                                  if ( arr1[i].tag = 1) then
                                       arr1[j].tag := 0;
                               end;
                              if ( arr1[i].tag = -1 ) then
                                   arr1[i].tag := 1;
                           end
                        else
                           begin
                             if (((arr1[i].kind_1 = arr1[j].kind_1) or
                                  (arr1[i].kind_1 = arr1[j].kind_2)) and
                                  (arr1[i].value = arr1[j].value) )then
                                   arr1[i].tag := 0;
                             if ( arr1[j].tag = -1 ) then
                                  arr1[j].tag := 1;
                           end;
                     end
                  else if (arr1[i].tag = -1) then
                           arr1[i].tag := 1;
                 end;
           end;
      end;
                                                  { EXTRACT RULES }
end;
```

```
{*************************************************************************
{*********************  PROCEDURE PRINT RULES  ***************************
{****                                                                  ****
{****  This procedure calls on the appropriate modules and prints    ****
{****  the headers and the output --> rules.                         ****
{*************************************************************************
procedure print_rules( dsub1_arr,dsub2_arr : value_array;
                       dinter1_arr,dinter2_arr : value_array);

var
   kind   : char;
   c_or_p : char;
   t_val  : real;
   ch     : char;
   size   : integer;
begin                                              { PRINT RULES }
   size := 8;
   kind_rules(kind);
   if (( kind <> 'q') or ( kind <> 'Q') ) then
      begin
         if ( ( kind = 'a') or (kind = 'A')) then
           begin
              t_val := 0.0;
              c_or_p := 'c';
              check_thresh(ch,c_or_p);
              if ( (ch = 'y') or (ch = 'Y') ) then
                begin
                   get_tval(t_val,c_or_p);
                   thresh_head(t_val);
                end;
              printall_head;
              certain_head;
              print_english(dec1_sub,c_or_p,t_val,kind,size);
              print_english(dec2_sub,c_or_p,t_val,kind,size);
              c_or_p := 'p';
              t_val := 0.0;
              check_thresh(ch,c_or_p);
              if ( (ch = 'y') or (ch = 'Y') ) then
                begin
                   get_tval(t_val,c_or_p);
                   thresh_head(t_val);
                end;
              possible_head;
              print_english(dec1_inter,c_or_p,t_val,kind,size);
              print_english(dec2_inter,c_or_p,t_val,kind,size);
           end
         else
           begin
              ch := ' ';
              t_val := 0.0;
              c_or_p := 'c';
              check_thresh(ch,c_or_p);
              if ( (ch = 'y') or (ch = 'Y') ) then
                begin
                   get_tval(t_val,c_or_p);
                   thresh_head(t_val);
                end;
              extract_rules(dec1_sub,c_or_p,t_val);
              extract_head;
              certain_extract_head;
              print_english(dec1_sub,c_or_p,t_val,kind,size);
              extract_rules(dec2_sub,c_or_p,t_val);
              print_english(dec2_sub,c_or_p,t_val,kind,size);
              t_val := 0.0;
              c_or_p := 'p';
```

```
                check_thresh(ch,c_or_p);
                if ( (ch = 'y') or (ch = 'Y') ) then
                  begin
                     get_tval(t_val,c_or_p);
                     thresh_head(t_val);
                  end;
                extract_rules(dec1_inter,c_or_p,t_val);
                possible_extract_head;
                print_english(dec1_inter,c_or_p,t_val,kind,size);
                extract_rules(dec2_inter,c_or_p,t_val);
                print_english(dec2_inter,c_or_p,t_val,kind,size);
          end;
     end;
end;                                                  { PRINT RULES }



{*************************************************************************}
{*********************  PROCEDURE ASSIGN VALUES  *********************}
{****                                                                 ****}
{****  This procedure assigns the values for the sample run.          ****}
{*************************************************************************}
Procedure assign_value( var arr1 : info );
                                                  { ASSIGN VALUES }
   begin
      arr1[1].condition_1.value_att1 := 0.3; arr1[1].condition_1.value_att2 :
.8;
      arr1[1].condition_2.value_att1 := 0.2; arr1[1].condition_2.value_att2 :
.9;
      arr1[1].decision.value_att1 := 0.3; arr1[1].decision.value_att2 := 0.6;

      arr1[2].condition_1.value_att1 := 0.4; arr1[2].condition_1.value_att2 :
.7;
      arr1[2].condition_2.value_att1 := 0.4; arr1[2].condition_2.value_att2 :
7;
      arr1[2].decision.value_att1 := 0.8; arr1[2].decision.value_att2 := 0.5;

      arr1[3].condition_1.value_att1 := 0.7; arr1[3].condition_1.value_att2
.4;
      arr1[3].condition_2.value_att1 := 0.6; arr1[3].condition_2.value_att2
.7;
      arr1[3].decision.value_att1 := 0.5; arr1[3].decision.value_att2 := 0.9

      arr1[4].condition_1.value_att1 := 0.8; arr1[4].condition_1.value_att.
.5;
      arr1[4].condition_2.value_att1 := 0.3; arr1[4].condition_2.value_att2
.8;
      arr1[4].decision.value_att1 := 0.7; arr1[4].decision.value_att2 := 0.3

      arr1[5].condition_1.value_att1 := 0.2; arr1[5].condition_1.value_att2
.7;
      arr1[5].condition_2.value_att1 := 0.2; arr1[5].condition_2.value_att2
.5;
      arr1[5].decision.value_att1 := 0.4; arr1[5].decision.value_att2 := 0.2

      arr1[6].condition_1.value_att1 := 0.9; arr1[6].condition_1.value_att2
.2;
      arr1[6].condition_2.value_att1 := 0.8; arr1[6].condition_2.value_att2
.2;
      arr1[6].decision.value_att1 := 0.7; arr1[6].decision.value_att2 := 0.8

      arr1[7].condition_1.value_att1 := 0.3; arr1[7].condition_1.value_att2
.6;
      arr1[7].condition_2.value_att1 := 0.7; arr1[7].condition_2.value_att2
.1;
      arr1[7].decision.value_att1 := 0.4; arr1[7].decision.value_att2 := 0.5
```

```
   end;                                                    { ASSIGN VALUES }


{*****************************************************************************
{************************  PROCEDURE EXAMPLE  ********************************
{****                                                                        ****
{****  This procedure assigns the conditional and decisional attributes      ****
{****  for the sample run.                                                   ****
{*****************************************************************************
procedure example(var con : cond_struct;
                  var dec : dec_struct;
                  var case_arr : info);

var
 i : integer;                                              { EXAMPLE }
begin
   initialize(con,dec);
   con.consider := 'tumor';
   dec.consider := 'disease';
   con.firattr  := 'size';
   con.secattr  := 'color';
   con.firattr_1 := 'large';
   con.firattr_2   := 'small';
   con.secattr_1  := 'red';
   con.secattr_2 := 'blue';
   dec.firattr  := 'Da';
   dec.secattr  := 'Db';
   for i := 1 to 8 do
      begin
         case_arr[i].condition_1.att1 := con.firattr_1;
         case_arr[i].condition_1.att2 := con.firattr_2;
         case_arr[i].condition_2.att1 := con.secattr_1;
         case_arr[i].condition_2.att2 := con.secattr_2;
         case_arr[i].decision.att1    := dec.firattr;
         case_arr[i].decision.att2    := dec.secattr;
      end;
   assign_value( case_arr );
end;                                                       { EXAMPLE }


{*****************************************************************************}
{************************  PROCEDURE MENU  ***********************************}
{****                                                                        ****}
{****            This procedure creates the main menu.                       ****}
{*****************************************************************************}
procedure menu (var sel : char );
   begin                                                   { MAIN }
      sel := '4';
    REPEAT
      writeln('    ************************************************');
      writeln('    ************************************************');
      writeln('    **                                            **');
      writeln('    **  WELCOME TO THE CULAS-WORM DECISION MAKER   **');
      writeln('    **                                            **');
      writeln('    ************************************************');
      writeln('    ************************************************');
      writeln;
      writeln;
      writeln;
      writeln('             Here are the choices      ');
      writeln('            ***********************    ');
      writeln('
      writeln('        1)  Show a sample run of this program  ');
      writeln('        2)  Run the program using your data    ');
```

```
      writeln('        3)  Quit                                          ');
      writeln(' ');
      writeln(' ');
      write('Please enter your choice 1, 2, or 3 : ');
      readln(sel);
    UNTIL ( ( sel = '1') or (sel = '2') or (sel = '3') );
  end;                                              { MAIN }


{*************************************************************************}
{*********************  PROCEDURE ASK SIT  ***************************}
{****                                                                  ****}
{****  This procedure asks the user if they would like to use the    ****}
{****  data for a different run.                                     ****}
{*************************************************************************}
procedure ask_sit(var choice : char;
                  var cont  : integer);
begin                                           { ASK SIT }
   choice := ' ';
   REPEAT
     writeln('Do you want to use the same attributes as previously used ');
     write( 'Please enter [y] or [n] ');
     readln(choice);
   UNTIL ( (choice = 'y')or(choice = 'Y') or ( choice = 'n') or
         (choice = 'N') );
end;                                            { ASK SIT }


{*************************************************************************}
{*********************  PROCEDURE ASK DATA KIND  *********************}
{****                                                                  ****}
{****  This procedure asks the user what type of data they will be   ****}
{****  using - real or fuzzy.                                        ****}
{*************************************************************************}
procedure ask_data_kind( var ch : char);
   begin                                        { ASK DATA KIND }
      ch := ' ';
      REPEAT
      writeln;
      writeln;
      writeln ('What kind of data do you want to use ?');
      writeln;
      writeln ('Real kind : eg. ( 20 30 92 ) ');
      writeln ('   or  ');
      writeln ('Fuzzy kind : (values between 0 and 1 : eg.(.1 .4 .7) ');
      writeln;
      write('Please enter [r] for real or [f] for fuzzy : ');
      readln(ch);
      UNTIL( (ch = 'r') or (ch='R')or(ch='f')or(ch='F') );
   end;                                         { ASK DATA KIND }


{*************************************************************************}
{*********************  PROCEDURE READ FUZZY  ************************}
{****                                                                  ****}
{****          This procedure reads in the fuzzy values.             ****}
{*************************************************************************}
procedure read_fuzzy(con : cond_struct;
                     dec : dec_struct;
                     var case_arr : info;
                     var n : integer);

var
  i  : integer;
```

```
        writeln;
        writeln('Would you like to see the data being used : ');
        write('Please enter [y] or [n] : ');
        readln(ch);
        until ( ( ch = 'y') or (ch = 'Y') or (ch = 'n') or (ch = 'N') );
    end;                                                    { ASK SEE }


{*************************************************************************}
{************************  PROCEDURE PRINT DATA  *************************}
{****                                                                  ****}
{****          This is the procedure to print the data.                ****}
{*************************************************************************}
procedure print_dat(arr1  : info;
                    con   : cond_struct;
                    dec   : dec_struct;
                     n    : integer );
var
   i : integer;
begin                                                   { PRINT DATA }
   writeln(' Data being used  ');
   writeln;
   writeln;
   writeln(' The attributes under consideration are ',con.consider,
           ' and ',dec.consider );
   writeln;
   writeln;
   for i := 1 to n do
      begin
         writeln(arr1[i].condition_1.att1:10,' = ':2,arr1[i].condition_1.value_a
tt1:2:1,
         ' ':10,arr1[i].condition_1.att2:10,' = ':2,arr1[i].condition_1.value_at
t2:2:1);
         writeln(arr1[i].condition_2.att1:10,' = ':2,arr1[i].condition_2.value_a
tt1:2:1,
         ' ':10,arr1[i].condition_2.att2:10,' = ':2,arr1[i].condition_2.value_at
t2:2:1);
         writeln(arr1[i].decision.att1:10,' = ':2,arr1[i].decision.value_att1:2:
1,
         ' ':10,arr1[i].decision.att2:10,' = ':2,arr1[i].decision.value_att2:2:1
);
         writeln;
         writeln;
      end;
   end;                                                 { PRINT DATA }


{*************************************************************************}
{*************************************************************************}
{*************************                       *************************}
{*************************      MAIN MODULE      *************************}
{*************************                       *************************}
{*************************************************************************}
{*************************************************************************}

BEGIN                                                   { MAIN }
   initialize(condition,decision);
   count := 0;
   m_ch := '4';
   repeat
   read_data := false;
   menu(m_ch);
```

```
    ch : char;
  begin
      ch := 'y';                                        { READ FUZZY }
      i := 1;
      while ( (ch = 'y') or (ch = 'Y') ) do
         begin
            case_arr[i].condition_1.att1 := con.firattr_1;
            case_arr[i].condition_1.att2 := con.firattr_2;
            case_arr[i].condition_2.att1 := con.secattr_1;
            case_arr[i].condition_2.att2 := con.secattr_2;
            case_arr[i].decision.att1 :=    dec.firattr;
            case_arr[i].decision.att2 :=    dec.secattr;
            write('Please enter a value for ',con.firattr_1, ' : ');
            readln(case_arr[i].condition_1.value_att1);
            write('Please enter a value for ',con.firattr_2, ' : ');
            readln(case_arr[i].condition_1.value_att2);
            write('Please enter a value for ',con.secattr_1, ' : ');
            readln(case_arr[i].condition_2.value_att1);
            write('Please enter a value for ',con.secattr_2, ' : ');
            readln(case_arr[i].condition_2.value_att2);
            write('Please enter a value for ',dec.firattr, ' : ');
            readln(case_arr[i].decision.value_att1);
            write('Please enter a value for ',dec.secattr, ' : ');
            readln(case_arr[i].decision.value_att2);
            i := i + 1;
         REPEAT
         writeln;
         write('Please enter [y] to input more data or [s] to stop : ');
         readln(ch);
         UNTIL ( (ch = 's') or (ch = 'S') or (ch ='y') or (ch = 'Y') );
   end;
      n := i-1;
     ch := ' ';
end;                                                { READ FUZZY }


{*************************************************************************}
{************************  PROCEDURE ASK VALUE  **************************}
{****                                                                     ****}
{****  This procedure asks the user if they want to use the same      ****}
{****  values as before for another run.                              ****}
{*************************************************************************}
procedure ask_value ( var ask_val : char);
   begin                                            { ASK VALUE }
     repeat
     writeln;
     writeln('Do you want to use the same values as previously used ');
     write('Please enter [y] for yes;  or [n] for no : ');
     readln( ask_val);
     until ( (ask_val = 'y') or (ask_val = 'Y') or (ask_val = 'n') or
          (ask_val = 'N') );
end;                                                { ASK VALUE }


{*************************************************************************}
{************************  PROCEDURE ASK SEE  **************************}
{****                                                                     ****}
{****  This procedure asks the user if they want to see the data      ****}
{****  that is being used.                                            ****}
{*************************************************************************}
procedure ask_see( var ch : char);
   begin                                            { ASK SEE }
      repeat
      writeln;
```

```
 if ( (m_ch = '1') or (m_ch = '2') ) then
   begin
      if ( m_ch = '1') then
         begin
            no_of_cases := 7;
            example(condition,decision,cases);
            ask_see(see_ch);
            if ( ( see_ch = 'y') or (see_ch = 'Y') ) then
            print_dat(cases,condition,decision,no_of_cases);
         end
else if (m_ch = '2') then
   begin
      if ( count = 0) then
        begin
           read_data := true;
           count := count + 1;
           initialize(condition,decision);
           read_situation(condition,decision);
        end
      else
        begin
           ask_sit(sit_ch,count);
           if ( (sit_ch = 'n') or (sit_ch = 'N') ) then
             begin
                read_data := true;
                initialize(condition,decision);
                read_situation(condition,decision)
             end
           else
              begin
                 ask_value(val_ch);
                 if ( (val_ch = 'n') or (val_ch = 'N') ) then
                 read_data := true;
              end;
        end;
 if read_data then
    begin
       ask_data_kind(d_ch);
       if ( (d_ch = 'r') or (d_ch = 'R') ) then
       read_real(condition,decision,cases,no_of_cases)
      else
       read_fuzzy(condition,decision,cases,no_of_cases);
       ask_see(see_ch);
       if ( ( see_ch = 'y') or (see_ch = 'Y') ) then
       print_dat(cases,condition,decision,no_of_cases);
    end;
end;

create_cond1_fuzzy_sets(cases,no_of_cases,cond1_att1_arr,cond1_att2_arr);
create_cond2_fuzzy_sets(cases,no_of_cases,cond2_att1_arr,cond2_att2_arr);
create_decision_fuzzy_sets(cases,no_of_cases,dec_att1_arr,dec_att2_arr);
init(dec1_sub);
init(dec2_sub);
init(dec1_inter);
init(dec2_inter);
set_cond(condition,dec1_sub);
set_cond(condition,dec2_sub);
set_cond(condition,dec1_inter);
set_cond(condition,dec2_inter);
set_dec1(decision,dec1_sub);
set_dec1(decision,dec1_inter);
set_dec2(decision,dec2_sub);
set_dec2(decision,dec2_inter);
value_sub(no_of_cases,cond1_att1_arr,cond1_att2_arr,cond2_att1_arr,
               cond2_att2_arr,dec_att1_arr,dec1_sub);
value_sub(no_of_cases,cond1_att1_arr,cond1_att2_arr,cond2_att1_arr,
```

```
                    cond2_att2_arr,dec_att2_arr,dec2_sub);
    value_inter(no_of_cases,cond1_att1_arr,cond1_att2_arr,cond2_att1_arr,
                    cond2_att2_arr,dec_att1_arr,dec1_inter);
    value_inter(no_of_cases,cond1_att1_arr,cond1_att2_arr,cond2_att1_arr,
                    cond2_att2_arr,dec_att2_arr,dec2_inter);
    print_rules(dec1_sub,dec2_sub,dec1_inter,dec2_inter);
end;
  until (m_ch = '3');
end.                                                    { MAIN }
```