Measuring Uncertainty by Extracting Fuzzy Rules Using Rough Sets

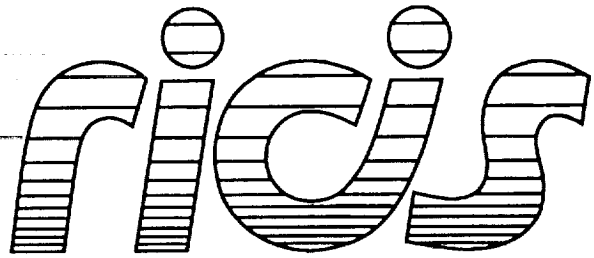
Jeffrey A. Worm

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.

Measuring Uncertainty by Extracting Fuzzy Rules Using Rough Sets

RICIS Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Jeffrey A. Worm 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.

MEASURING UNCERTAINTY BY EXTRACTING FUZZY RULES USING ROUGH SETS

Prepared by

Jeffrey A. Worm

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	_____
Dr. Kenneth Oberhoff	Member	_____
Dr. Ananda Gunawardena	Member	_____
Dr. Robert Lea	Member	_____
Dr. Richard A. Alo'	Chairman	_____

Abstract

Despite the advancements in the computer industry in the past thirty years, there is still one major deficiency. Computers are not designed to handle terms where uncertainty is present. To deal with uncertainty, techniques other than classical logic must be developed. This paper examines the methods of statistical analysis, the Dempster-Shafer theory, rough set theory, and fuzzy set theory to solve this problem. The fundamentals of these theories are combined to possibly provide the optimal solution. By incorporating principles from these theories, a decision-making process may be simulated by extracting two sets of fuzzy rules: certain rules and possible rules. From these rules a corresponding measure of how much we believe these rules is constructed. From this, the idea of how much a fuzzy diagnosis is definable in terms of a set of fuzzy attributes is studied.

Acknowledgements

I would like to express my thanks to Dr. Andre' de Korvin for his very helpful advice throughout the semester. It is his paper, *Extracting Fuzzy Rules under Uncertainty and Measuring Definibility using Rough Sets*, which provided the cornerstone for my research and the blueprint for my software.

I would also like to extend my thanks to Dr. Kenneth Oberhoff for his helpful insights in all phases of software design and programming algorithms.

Finally, I also extend my thanks to Dr. Richard A. Alo' for his constructive comments throughout the semester regarding presentations and all initial drafts of this document.

MEASURING UNCERTAINTY BY EXTRACTING FUZZY RULES USING ROUGH SETS

TABLE OF CONTENTS

INTRODUCTION

Computers have progressed so much over the past thirty years that it is now hard to imagine life without them. They have become smaller, faster, and less expensive. Similarly, the applications we use them for have grown exponentially. If the auto industry had done what the computer industry has done in this time, a Rolls-Royce would cost a couple of dollars and might get a million miles per gallon.

An important development of this progression is the computer's ability to refine and expedite the decision-making process. One can enter raw data as input and receive the output in an organized, logical form. This manipulated form may then be used to help facilitate some type of decision by the user. It is also possible for a computer program to have a built in "thinking" function which requires no help from the user in order to formulate a decision. A decision may be automatically made by the computer, solely on the output and any preset conditions of the output. This may be achieved through a series of If-then-else statements, for

example.

A program which can perform these simple functions is possible through knowledge acquisitions using examples. Through repetition, one may learn to associate certain factors to form a decision. Ideally, the decisions will always be the same if the corresponding factors are always the same. For example, if a person sees lightning and hears thunder, they may assume it is raining close by from some similar experiences in the past. Again, this is under "ideal" circumstances; the person is positive they see lightning and positive they hear thunder. Unfortunately, "ideal" circumstances are not always present.

As amazing as the progression of computers has been, there is a noticeable deficiency: computers are not designed to manipulate data where uncertainty is present. Uncertainty may arise in many different ways. It may be brought about by ambiguous terms used to describe a certain situation. It may also be caused by scepticism of rules used to describe a course of action, or by missing or erroneous data. To handle uncertainty, methods other than classical logic must be developed. One possible solution to this is to use fuzzy set theory to extract rules.

In ordinary set theory, an element is either in or out of the set. In **fuzzy set** theory, however, an approximation is used to determine the

degree to which an element is in the set. This is due to the fact that subjective terms are often used to describe a condition. Fuzzy set theory allows for a fraction of an element to be in the set. From these fuzzy sets, one can extract two sets of fuzzy rules: certain rules and possible rules. Basically, the certain rules are formed by taking the minimum of the union of two fuzzy sets. Conversely, the possible rules are formed by taking the maximum of the intersection of two fuzzy sets.

A possible solution to deal with uncertainty is in learning from examples. An effective method to acquire knowledge through examples is rough sets. **Rough sets** are the group of sets having the same upper and lower approximations. As in fuzzy set theory, possible and certain rules are extracted. In rough sets, these rules are generated by the upper and lower approximations. These qualities are similar to the inner and outer reductions of Dempster-Shafer theory. The theory and notation of upper and lower approximations is discussed in more detail in section 2.1. The attributes of the conditions are assigned values and a measure of how much these attributes determine the diagnosis is established. However, the values of these attributes require some judgement for their determination. Similary, the diagnosis is often not of "pure" type, but a combination which is reflective of fuzzy sets.

Combining these two methods of fuzzy set and rough set theories, as well as the principles of Dempster-Shafer theory, provides a possible optimal solution for dealing with uncertainty. By integrating these two methods, we can produce a set of certain rules and possible rules and determine a measure of belief associated with these rules. These rules allow a foundation for dealing with uncertainty in the decision-making process.

- Section 1 -

Uncertainty

1.1 Uncertainty

The previously referenced computer program took certain variables and "crunched" them up to come to a certain decision. A major question that arises is, "How does one deal with uncertainty?". Uncertainty may arise in many different situations. It may be caused by the ambiguity in the terms used to describe a specific situation, or it may be caused by the skepticism of rules used to describe a course of action. Uncertainty may also be caused by inconsistencies in data, or simply by missing or erroneous data.

To understand what is meant by ambiguity of terms, one must realize that different people may associate different meanings or values for the same term(s). To illustrate this, one cannot put a set value on "very rich" or "moderately rich" because these are subjective terms. One person's definition may be quite different from another's. For this reason, descriptive terms may contain some degree of ambiguity, and therefore some degree of uncertainty.

Uncertainty caused by the skepticism of rules may be attributed to an underlying doubt one may have regarding a situation. Occasionally, all factors may point towards a certain decision, but one's "gut feeling" produces a degree of doubt toward that decision. Whether these doubts are warranted or not, they must be taken into account when we refer to uncertainty. For these doubts may influence one's future decisions on similar situations.

Clearly, any missing or erroneous data will lead to uncertainty. Unfortunately, it is not always obvious when data is wrong. A strong characteristic of erroneous data is inconsistencies. In other words, if the same data produces conflicting outcomes, uncertainty is present. To illustrate this, the table below represents how a decision-maker may make an inconsistent decision based on a couple of pieces of data. In this example, Case X_2 and Case X_3 have the same data, yet different decisions. This shows that uncertainty exists somewhere in this decision-making process.

	CONDITIONS		
CASE	DATA1	DATA2	DECISION
X_1	W	Y	A
X_2	X	Z	B
X_3	X	Z	A
X_4	X	Y	B
X_5	W	Y	A

1.2 Techniques to Combat Uncertainty

1.2.1 Statistics

To deal with uncertainty, techniques other than classical logic need to be developed. The most useful tool for handling probability is statistics, or statistical analysis. **Statistical analysis** is concerned with the collection, organization, and interpretation of data according to well-defined procedures. Observations are made and converted into numerical form. The numbers are manipulated and organized, with the results interpreted and translated back into a way one may understand.

Statistical analysis allows for the reduction of data. Large masses of unorganized numbers may be characterized into smaller sets that describe the original observations without sacrificing critical information. The second major role lies in its use as an inferential measuring tool. In other words, it provides procedures for stating the degree of confidence one may have in the accuracy of the measurements one makes. Finally, statistical analysis allows one to make distinctions about relationships that exist between and among sets of observations. Does knowledge about one set of data allow us to infer or predict characterisics about another set of data?

Stastistical analysis does, however, have some deficiencies. Data

reduction may lead to the sacrificing of detail. The inferential measuring tool statistical analysis provides is useful, but all measurements are subject to error. Furthermore, sometimes one may strive to find a connection between two sets so much that a connection is unjustifiably made.

Though statistics is a useful method for handling uncertainty, it provides only a foundation for the problem of knowledge acquisition under uncertainty. Three theories which are better suited to handle this problem are: Dempster-Shafer Theory, fuzzy set theory, and rough set theory.

1.2.2 Dempster-Shafer Theory

The Dempster-Shafer Theory is a theory of evidence and probable reasoning. It is a theory of evidence because it deals with weights of evidence and with numerical degrees of support based on evidence. It is a theory of probable reasoning because it focuses on the combination of evidence, more specifically, the combination of belief functions.

The theory begins with the idea of using a number between zero and one to indicate the degree of belief one should assign for inclusion on the basis of the evidence. Its focus lies in the combination of degrees of belief based on one body of evidence with those based on an entirely

distinct body of evidence. This combination of belief functions is the heart of the Dempster-Shafer Theory. Given several belief functions based on distinct bodies of evidence, this theory enables one to compute a new belief function based on the combined evidence.

The main connection Dempster-Shafer has to the other theories is the concepts of inner and outer reductions. As will be shown in the discussion concerning rough sets (section 2.1), this concept is almost identical to the lower and upper approximations of rough sets. In inner reduction, denoted by $\underline{\Theta}(A)$, is the largest subset that implies A. The outer reduction, denoted by $\overline{\Theta}(A)$, is the smallest subset that is implied by A.

Another connection the Dempster-Shafer theory holds to that of rough set theory is its belief and plausibility theorems. Given a belief function (Bel) committed to the subset A one is given Bel(A). This function allows one to study the extent to which the evidence supports the negation of A, i.e. $\neg$A, --> Bel(A). The quality [1 - Bel(A)] expresses the plausibility of A; the extent to which the evidence allows one to fail. These theories are very similar to rough set theory.

1.2.3 Fuzzy Set Theory

Perhaps the most useful tool when dealing with uncertainty is fuzzy set theory. This theory is the most practical where ambiguous terms are

present. To get a complete understanding of this theory, one must first backtrack to ordinary set theory. All branches of mathematics are developed, consciously or unconsciously, in set theory or some part of it. It is, therefore, an important concept to grasp. A set is a collection of things (called elements or members), the collection being regarded as a single object. An item is either in the set or it is not. This property is referred to as inclusion.

In **fuzzy set** theory, however, an approximation is used to determine the degree to which an element is in the set. Such concepts as inclusion or set equality may seem too strict. Usually, the structures embedded in fuzzy set theories are less rich than the boolean lattice of ordinary set theory. Unlike ordinary set theory, one cannot determine the cardinality, or size, in fuzzy set theory. One cannot compute an accurate union or intersection of two fuzzy sets because the elements are estimates of inclusion, not "crisp" values.

If the value of a set is allowed to be the real interval [0,1], A is called a fuzzy set. The grade of membership of an element, x, in A is $\mu_A(x)$. The closer the value of $\mu_A(x)$ is to 1, the more x belongs to A. Similarly, the lower the value of $\mu_A(x)$, the less x belongs to A. Clearly, A is a subset of x that has no crisp boundary. By using fuzzy set theory, one

must approximate the value of inclusion an element has in a set.

Earlier the question was raised, "What is the difference between 'very rich' and 'moderately rich'?". Fuzzy set theory could approximate a person worth $X to be .4/very rich and .8/moderately rich. Because of the ambiguity of the term "rich", one needs to approximate the value of the person for "very rich" and "moderately rich". It might be observed that, for the decision-maker assigning the values, the person falls into the category of "moderately rich" more than "very rich". For this reason, the decision-maker puts more "weight" on the term "moderately rich". The person lies within the set of "moderately rich" more than the set of "very rich". Hence, they are assigned those corresponding values.

A problem one may encounter using this theory is the fact that the decision-maker assigns these values. Obviously, not all people have the same pre-conceived meanings for terms such as "very rich" or "extremely tall". The approximations one person gives may be completely different from the approximations of someone else. For example, a small boy may see a man 5'9" as "very" tall. Conversely, a professional basketball player might see the same person as "average" height. It is best to keep this in mind, because it can easily influence the decisions.

1.2.4 Rough Sets

As was stated earlier, the most traditional way of acquiring knowledge is based on learning from examples. An another effective tool of inferring knowledge from examples is rough sets. **Rough sets** are the family of sets having the same lower and upper approximations.

Let U be a non-empty set, call 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.

Two more concepts, known as the lower approximation and upper approximation of X in A are examined later. Basically, the lower approximation of X in A is the greatest definable set in A, contained in X. The upper approximation of X in A is the least definable set in A containing X. These concepts correspond to the inner and outer reductions from Dempster-Shafer Theory, also examined later. A rough set in A is the family of all subsets of U having the same lower and upper approximations in A. These concepts are examined in more detail in

section 2.1.

There are essential connections between rough set theory and Dempster-Shafer theory. For example, the lower and upper approximations of rough set theory exist under the names of inner and outer reductions, respectively. Similary, the qualities of lower and upper approximations of rough set theory are the belief and plausibility functions, respectively, of Dempster-Shafer theory.

The main difference between rough set theory and the Dempster-Shafer Theory is in the emphasis: Dempster-Shafer Theory uses belief functions as a main tool, while rough set theory makes use of the family of all sets with common lower and upper approximations. The main advantage of rough set theory is that it does not need any preliminary or additional information about data.

1.3 The Proposed Solution

The main purpose of this work is to study the setting described before where a decision-maker is faced with uncertain (i.e. fuzzy) conditions and makes a fuzzy decision which might be strongly or weakly based on these symptoms. Here, the techniques or fuzzy set theory and rough sets will be incorporated to attempt to provide the optimal solution

of measuring uncertainty. From the conditions and decisions, one will find that fuzzy rules may be extracted. In fact, one may extract two sets of rules: certain rules and possible rules. One may also determine a measure of how much they believe in these rules.

The main body of this work is examined in detail in Section 2. The basic notations and results necessary to fully understand these concepts are discussed here. Section 3 offers a detailed example of these concepts at work. It provides an everyday application, as well as an opportunity to see how these principles are incorportated. The software specifications of this product, which simulates the basic ideas set forth here, is available in Section 4. The coding of this program may be found in Appendix A.

- SECTION 2 -

Rough Set theory vs. Fuzzy Set theory

As stated in Section 1, I believe the optimal solution for knowledge acquisition under uncertainty lies within the combination of fuzzy set and rough set theories. By integrating the fundamentals of these theories, I hope to measure and, where possible, minimize the degree of uncertainty. To best understand how the concepts of fuzzy sets and rough sets are to be incorporated, it is important to first grasp the main principles of these two theories.

2.1 Rough Sets - A Closer Examination

Let *U* be the universe, *R* an equivalence relation on *U*,and *X* any subset of *U*. If [*X*] denotes the equivalence class of *X* relative to *R*, we can then define the foundation of rough sets. This is called the lower and upper approximations of *X* and is denoted, respectively, by:

$$\underline{R}(X) = (X \in U / [X] \subset X) \qquad \textit{and}$$

$$\overline{R}(X) = (X \in U / [X] \cap X \neq \varnothing).$$

Once again, rough sets are the family of all subsets in *U* having the same

upper and lower approximations.

To examine these upper and lower approximations closer, we define an information system as the quadruple (U,Q,V,τ) where $Q = C \cup D$ and $C \cap D = \varnothing$. The set C stands for the set of conditions, and D is the set of decisions. We assume that C is equal to the set of attributes, Q. The set V stands for value and τ is a function from U x Q into V where $\tau(u,q)$ denotes the value of attribute q for element u. For example, the pulse rate q of patient u. The set C produces an equivalence on *U* by partitioning *U* into sets over which all attributes are constant. A rough set is classified by properties of its lower and upper approximations. The set is called roughly C-definable if:

$$\underline{R}(X) \neq \varnothing \text{ and } \overline{R}(X) \neq U.$$

The set is internally C-undefinable if:

$$\underline{R}(X) = \varnothing \text{ and } \overline{R}(X) \neq U.$$

The set is externally C-undefinable if:

$$\underline{R}(X) \neq \varnothing \text{ and } \overline{R}(X) = U.$$

The set is totally C-undefinable if:

$$\underline{R}(X) = \varnothing \text{ and } \overline{R}(X) = U.$$

To illustrate some of these ideas, the table previously referenced is examined again. For this example, Decision 'B' denotes sickness. The conditions produce a partition on $\{X_1,X_2,X_3,X_4,X_5\}$, namely { $\{X_1,X_5\}$, $\{X_2,X_3\}$ $\{X_4\}$ }. The decision-maker defines "sick" people by $X = \{X_2,X_4\}$. Thus, the lower and upper

	CONDITIONS		
CASE	DATA1	DATA2	DECISION
X_1	W	Y	A
X_2	X	Z	B
X_3	X	Z	A
X_4	X	Y	B
X_5	W	Y	A

approximations are $\underline{R}(X) = \{X_4\}$ and $\overline{R}(X) = \{X_2,X_3,X_4\}$. These upper and lower approximations are used to extract the certain and possible fuzzy rules, respectively. For this particular example, the set $\{X_4\}$ representsthe set of people who are sick for certain, while the set $\{X_2,X_3,X_4\}$ represents the set of people who possibly could be sick. For the case X_3, decision 'B' could lead to the question, "Why is that case possibly sick?". The reason is because of the inconsistencies created by X_2 and X_3, and the fact that the same conditions lead to sickness (decision 'A') for X_2. Because $\underline{R}(X) \neq \varnothing$ and $\overline{R}(X) \neq U$, X is roughly C-definable.

For an internally C-undefinable set X in S we can not say with

certainty that any $x \in U$ is a member of X. To demonstrate this case, we assume an additional case, X_6, is added to the previous table. We now have:

	CONDITIONS		
CASE	DATA1	DATA2	DECISIONS
X_1	W	Y	A
X_2	X	Z	B
X_3	X	Z	A
X_4	X	Y	B
X_5	W	Y	A
X_6	X	Y	A

This creates a new partition of $X = \{ \{X_1,X_5\} , \{X_2,X_3\} , \{X_4,X_6\} \}$. We now have no certain cases of sickness, because for every case corresponding to sickness, there is an inconsistency to match it. Therefore, $\underline{R}(X) = \emptyset$ and $\overline{R}(X) = \{X_2,X_3,X_4,X_6\}$.

For an externally C-undefinable set X in S we can not exclude any $x \in U$ being possibly a member of X. If case X_1 were changed to produce this:

	CONDITIONS		
CASE	DATA1	DATA2	DECISION
X_1	W	Y	B
X_2	X	Z	B
X_3	X	Z	A
X_4	X	Y	B
X_5	W	Y	A
X_6	X	Y	A

The new partition of { $\{X_1\}$, $\{X_2,X_3\}$, $\{X_4,X_6\}$, $\{X_5\}$ } would be created. This would then make $\underline{R}(X) = \{\emptyset\}$ and $\overline{R}(X) = \{X_1,X_2,X_3,X_4,X_5,X_6\}$ or $R(X) = U$. Thus,we could say all six cases are possibly sick.

The difference between the lower and upper approximations may be attributed to the presence of inconsistencies. If it were not for the inconsistencies, the decision-maker's opinion would be in line with the upper and lower approximations produced by C. Therefore, X would be totally expressible in terms of C. It is this difference between $\overline{R}(X)$ and $\underline{R}(X)$ that offers a measure of how well the diagnosis of the decision-maker follows the conditions. If the decision-maker is an "expert", the difference between the lower and upper approximations gives one a measure of how good conditions C are to determine the diagnosis. In other words, the more we trust the decision-maker, the more we believe how the conditions determine the diagnosis. Moreover, it is these lower and upper approximations which generate the rules that will be used as the basis for the decision-making process. These generated rules, called the certain and possible rules, will be examined closer in Section 2.2.

Unfortunately, there may be uncertainty in the conditions, as well as the diagnosis. The conditions and the diagnosis rarely partition the universe into "crisp" sets. This is due to the fact that most of the values

of attributes are descriptive, and thus subjective terms. It is this that leads to the "fuzziness" of the conditions/diagnosis when trying to define the terms. This "fuzziness" can lead to overlapping, therefore rendering crisp partitions nearly impossible. At best one hopes the terms can be partitioned with as little overlapping as possible.

2.2 Fuzzy Sets - A Closer Examination

For all decision-making processes, it is the rules which guides one towards a decision. Decision-making under uncertainty is no different. The problem lies within determining these rules. As stated in Section 2.1, the upper and lower approximations generate possible and certain rules. It is Fuzzy Set theory which allows one to extract these fuzzy rules.

2.2.1 Functions of Fuzzy Set Properties

To understand how these rules are extracted, one must first be familiar with the notation. A fuzzy subset A of U is defined by the function: $\mu_A : U \text{ -----> } [0,1]$.

This simply states that the values of the fuzzy subset A fall between 0 and 1. If A and B are fuzzy subsets, the properties $A \cap B$, $A \cup B$, and $\neg A$ are defined by the functions: $\text{Min}\{\mu_A(x),\mu_B(x)\}$, $\text{Max}\{\mu_A(x),\mu_B(x)\}$, and $1- \mu_A(x)$, respectively. The property $\neg A \cup B$ corresponds to the function

Max{1-A(x), B(x)}. These computed values are the foundation for extracting the rules. Therefore, it is very important to understand what is meant by the notation.

The first function, $\text{Min}\{\mu_A(x),\mu_B(x)\}$, is computed by matching up the corresponding elements of the fuzzy subsets and taking the minimum (in value) of the two. For example, given the two fuzzy subsets:

$$A = (.3, .4, .7, .8, .6, .1) \quad \textit{and}$$
$$B = (.6, .2, .4, .3, .5, .4)$$

One can compute Min(A,B) = (.3, .2, .4, .3, .5, .1). The second function, $\text{Max}\{\mu_A(x),\mu_B(x)\}$, is similar in computation to the first. Instead of taking the minimum of the two, one takes the maximum, or greatest in value. Using the two previous subsets of A and B, one can compute Max (A,B) = (.6, .4, .7, .8, .6, .4). The third function, $1 - \mu_A(x)$, is computed by taking one(1) minus the values of the fuzzy subset. Again, using the previous subset A, one can compute 1-A = (.7, .6, .3, .2, .4, .9). The last function, Max{1-A(x), B(x)}, is simply a combination of the second and third functions. First, one computes 1-A(x) then compares that to B(x), taking the maximum of the two. For example,

$$\text{Max}\{1\text{-A, B}\} = \text{Max}\{(.7, .6, .3, .2, .4, .9),(.6, .2, .4, .3, .5, .4)\}$$
$$\downarrow \qquad \downarrow \qquad \downarrow$$
$$\text{Max}\{1\text{-A, B}\} = (.7, .6, .4, .3, .5, .9).$$

2.3 Establishing Certain and Possible Rules

Now that the fundamental properties (and corresponding notation) have been explained, we can define two functions of major importance to this work. These two functions are on pairs of fuzzy sets and allow us to extract the rules. We assume here that 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. This function computes the rules generated by certainty and is defined as:

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

The function J(A#B) measures the degree to which A intersects B. This function computes the rules generated by possibility and is defined by:

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

The function $I(A \subset B)$ is computed by first finding Max{1-A(x), B(x)}, then taking the minimum term. For the previous fuzzy subset examples of A and B, we found the Max{1-A, B} = (.7, .6, .4, .3, .5, .9). Since the minimum term is .3, $I(A \subset B) = .3$. The function J(A#B) is computed by first finding Min{A(x), B(x)}, then taking the greatest (in value) term. Again, using A and B we found Min{A,B} = (.3, .2, .4, .3, .5, .1). Since the maximum term is .5, J(A#B) = .5.

For the example used in this work, we assume the decision-maker is faced with different conditions, or attributes, and makes a decision based on the values of these attributes. To provide a more concise explanation of this work, we will limit the number of possible decisions to two (2). Similarly, we will limit the description an attribute may have to two (2). For example, size can only be measured as a degree of large and small. These limitations are made to explain when to compute the $I(A \subset B)$ and $J(A\#B)$ values.

For the functions of $I(A \subset B)$ and $J(A\#B)$, A denotes the descriptions of the attributes, while B denotes the possible decisions. For each description, we must measure the degree to which it is included in decision 'A' as well as in decision 'B'. In addition to this, we also measure the degrees of intersections of the descriptions for each decision. For example, if we have attribute-1 with descriptions of 'W' and 'X', attribute-2 with descriptions of 'Y' and 'Z', and possible decisions of 'A' and 'B', we would need to compute all of the following:

$I(W \subset A)$	$I(Y \subset A)$	$I(W \cap Y \subset A)$	$I(W \cap Z \subset A)$
$I(W \subset B)$	$I(Y \subset B)$	$I(W \cap Y \subset B)$	$I(W \cap Z \subset B)$
$I(X \subset A)$	$I(Z \subset A)$	$I(X \cap Y \subset A)$	$I(X \cap Z \subset A)$
$I(X \subset B)$	$I(Z \subset B)$	$I(X \cap Y \subset B)$	$I(X \cap Z \subset B)$

J(W#A)	J(Y#A)	J(W∩Y#A)	J(W∩Z#A)
J(W#B)	J(Y#B)	J(W∩Y#B)	J(W∩Z#B)
J(X#A)	J(Z#A)	J(X∩Y#A)	J(X∩Z#A)
J(X#B)	J(Z#B)	J(X∩Y#B)	J(X∩Z#B)

2.3.1 Threshold Values

As one can see, this leads to large numbers of rules. For this case, we would have 32 rules: 16 certain rules and 16 possible rules. If we had 3 attributes with 2 descriptions each, the number of rules would increase to 104 rules. It is therefore essential to establish a "threshold" value, denoted by α, for which we may ignore all rules falling below this value. Actually, we need two of these values: one for the certain rules and one for the possible rules. The decision-maker may or may not set these two equal. The higher we set the threshold, the higher the belief we have for the rules which factor above it. Unfortunately, there is a trade-off; for the higher the threshold, the more rules we ignore. Ideally, the solution to this trade-off is to allow the decision-maker to interactively change the threshold values as they see fit. By allowing this interactive changing, it also provides somewhat of a sensitivity analysis. The decision-maker can immediately see which rules are affected by the changing threshold value. Another reason to promote interactive changing of the threshold is that the value of α is very much problem dependent. A value of $\alpha = .5$ might be

appropriate for one problem, but irrelevant for another. The decision-maker may adjust the value till it is set at the most appropriate level.

2.3.2 Extracting Possible and Certain Rules

Once the threshold value has been established, it is time to extract the rules. All rules (values of I and J) which fall below the threshold value are immediately eliminated. To further eliminate rules, we have certain provisions. First, all rules with unique I and J values are kept. Second, if more than one rule has identical I values, we keep (extract) the "smaller" in terms of attributes. For example, if we were to obtain the following certain rules:

If **W** then **A** is present .6	(1)
If **W** and **Y** then **A** is present .6	(2)
If **W** and **Z**then **A** is present .6	(3)

we would keep rule (1) because rules (2) and (3) offer no significant data. Conversely, if these three rules were computed using J values, thus making them possible rules:

If **W** then **A** is possible .6	(4)
If **W** and **Y** then **A** is possible .6	(5)
If **W** and **Z** then **A** is possible .6	(6)

we would extract rules (5) and (6). This is because rules (5) and (6) imply the possibility of rule (4).

The concepts discussed up to this point are represented in an example in section 3.

2.4 Definibility of Terms

Now that all the certain and possible rules are extracted, we can measure the definibility of terms. The goal of this is to define the terms in the decisions as a function of the terms in the conditions. How well this can be accomplished is a function of how much the decision follows the conditions.

Let $\{Q_i\}$ be a finite family of fuzzy sets. This family of sets does not necessarily form a partition on the universal set. Let A be a fuzzy set. A lower approximation of A through $\{Q_i\}$, produces the fuzzy set:

$$\underline{R}(A) = \mathbf{U}\ I(Q_i \subset A)\ Q_i.$$

Here, **U** denotes the union of fuzzy sets, and $I(Q_i \subset A)\ Q_i$ denotes the fuzzy set obtained by multiplying the components of Q_i by $I(Q_i \subset A)$. Therefore, if Q_i is very much a subset of A, $I(Q_i \subset A)\ Q_i$ is close to the whole set Q_i. Conversely, if $I(Q_i \subset A)$ is small, so is the contribution of Q_i to $\underline{R}(A)$. Similarly, we can define an upper approximation of A through $\{Q_i\}$ by:

$$\overline{R}(A) = \mathbf{U}\ J(Q_i \# A)\ Q_i.$$

In the special cases where all the sets are crisp, and $\{Q_i\}$ denotes a partition generated by an equivalence relation R, then the lower approximation is defined as:

$$\underline{R}(A) = \{X / [X] \subset A\},$$

and the upper approximation is defined as:

$$\overline{R}(A) = \{X \,/\, [X] \cap A \neq \varnothing\}.$$

One can therefore see that in this crisp case:

$$\underline{R}(A) \subset A \subset \overline{R}(A).$$

One should not, however, expect these inclusions to hold in the fuzzy case because boundaries of the relevant sets are poorly-defined.

It is important, however, to realize that even if the decisions completely follow the conditions, the rules generated are not necessarily 100% accurate. To illustrate this, we can substitute tumor color with shirt color when dealing with medically-based decisions. We might find that for 100 cases of cancer, all cases involved blue shirts. Does this mean blue shirts may promote cancer? The answer is, or course, no, but it forces the decision-maker to choose the conditional attributes carefully. The more the decision-maker knows the conditions influence the decisions, the more accurate the generated rules will be.

- SECTION 3 -

Application: Tumor Diagnosis

As stated earlier in this paper, knowledge aquisition is best accomplished by looking at examples. It is therefore important to provide an example of the concepts discussed in section 2. By examining an application, these concepts should become clearer.

The analogy to be used here is that of a doctor (decision-maker) examining the characteristics (attributes) of tumors and rendering a diagnosis (decision). For this example, the attributes are size and color. Size can be described as large and small. Color will be limited to the descriptions of blue and red. The possible diagnosises will be either Disease 'D_A' and 'D_B'.

While examining seven patients, the following data is accumulated:

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

From this data, we can extract the fuzzy sets for each of the descriptions (of the attributes) and for each diagnosis. These fuzzy sets are:

FUZZY SETS:

$L = .3/P1 + .4/P2 + .7/P3 + .8/P4 + .2/P5 + .9/P6 + .3/P7$

$S = .8/P1 + .7/P2 + .4/P3 + .5/P4 + .7/P5 + .2/P6 + .6/P7$

$R = .2/P1 + .4/P2 + .6/P3 + .3/P4 + .2/P5 + .8/P6 + .7/P7$

$B = .9/P1 + .7/P2 + .7/P3 + .8/P4 + .5/P5 + .2/P6 + .1/P7$

$D_A = .3/P1 + .8/P2 + .5/P3 + .7/P4 + .4/P5 + .7/P6 + .4/P7$

$D_B = .6/P1 + .5/P2 + .9/P3 + .3/P4 + .2/P5 + .8/P6 + .5/P7$

Now that we have established the fuzzy sets, we can compute the values of $I(A \subset B)$ and $J(A \# B)$ to establish the rules. These values are as follows:

$I(L \subset D_A) = .5$	$I(L \subset D_B) = .3$	$I(S \subset D_A) = .3$
$I(S \subset D_B) = .3$	$I(R \subset D_A) = .4$	$I(R \subset D_B) = .5$
$I(B \subset D_A) = .3$	$I(B \subset D_B) = .3$	$I(L \cap R \subset D_A) = .5$
$I(L \cap R \subset D_B) = .6$	$I(L \cap B \subset D_A) = .5$	$I(L \cap B \subset D_B) = .3$
$I(S \cap R \subset D_A) = .4$	$I(S \cap R \subset D_B) = .5$	$I(S \cap B \subset D_A) = .3$
$I(S \cap B \subset D_B) = .5$	$J(L \# D_A) = .7$	$J(L \# D_B) = .8$
$J(S \# D_A) = .7$	$J(S \# D_B) = .6$	$J(R \# D_A) = .7$
$J(R \# D_B) = .8$	$J(B \# D_A) = .7$	$J(B \# D_B) = .7$
$J(L \cap R \# D_A) = .7$	$J(L \cap R \# D_B) = .8$	$J(L \cap B \# D_A) = .7$
$J(L \cap B \# D_B) = .7$	$J(S \cap R \# D_A) = .4$	$J(S \cap R \# D_B) = .5$
$J(S \cap B \# D_A) = .7$	$J(S \cap B \# D_B) = .6$	

Assuming that we have already established the threshold value (α) for certain rules to be $\alpha = .5$ and for possible rules to be $\alpha = .6$, we are left

with the following rules:

CERTAIN RULES:

If the tumor is large then D_A is present 0.5.

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

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

If the tumor is red then D_B is present 0.5.

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

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

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

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.

Finally, we extract the certain rules and possible rules in which to keep by using the theory explained in section 2.3. This leads to the following rules:

EXTRACTED CERTAIN RULES:

If the tumor is large then D_A is present 0.5.

If the tumor is red then D_B is present 0.5.

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

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

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.

- SECTION 4 -

Software Specifications

4.1 Software Specifications

The following is an attempt to describe the requirements specifications for the software to be developed for partial fulfillment of the senior project (CS 4395). The software should be designed to simulate the main ideas in Dr. Andre' de Korvin's paper, "*Extracting fuzzy rules under uncertainty and measuring definibility using rough sets.*"

As in all good software design, the software should be above all user-friendly. It should be designed to allow a user to "walk-through" the system. This can be achieved through screen messages at every step and error messages when appropriate (improper data entry). The software should also be modifiable so that it may be expanded in the future. This can be achieved through well-documented modules. The software should also be efficient and reliable.

These are the goals of every software system. The following is a list of the functions, goals, and constraints of this particular system. In some instances, examples are used to better explain the concepts.

4.1.1 Input

The user should be able to:

A. Enter any number of attributes.
   The paper uses two, for example: size and color. The user should also be allowed to have any number of descriptions for each attribute. The paper describes size with values of large and small. The user should also be able to use medium.

B. Enter data in any numeric form.
   1) The form the data is entered in the paper is in "fuzzy form", where all values are between 0 and 1. The software should certainly be able to manipulate data which is entered in this form. In addition, the user should be able to enter "real data". For example, given the following numbers:

10	40	5	50
15	27	80	25
60	35	55	33

   The software should be able to convert 55 to
   55 -> .3/Low + .7/High
   2) The user should also be able to set the boundaries for the data to be entered. Using the numbers from above, the user may wish to declare 10 as the bottom and 75 as the ceiling. If the number 5 is entered as data, it should be converted to: 5 -> 1/Low + 0/High. Likewise, 80 would be converted to: 80 -> 0/Low + 1/High.
   The user should be able to arbitrarily set these boundaries as well as change them between applications.

C. Set the two threshold values (one for the certain rules, one for the possible rules).
   1) The user should be able to interactively change the threshold to compare the changes, i.e. the rules the changes affect.
   2) Software should produce an error message for a threshold value greater than 1 or less than 0.

4.1.2 Functions and Calculations

The software should be able to:

A. Convert the inputed data into the "fuzzy sets" that are used as the basis for all fuctions and calculations.

B. Measure the degree to which a set, A, is included in another, B: {I(A∩B)}. This calculation is used to determine the certain rules.

C. Measure the degree to which a set, A, intersects another, B: {J(A#B)}. This calculation is used to determine the possible rules.

D. Compare the values of I(A∩B), for various A's and B's, with the threshold value for certain rules and disregard all values of I(A∩B) which fall below the threshold. Similarly, all values of J(A#B) should be compared to the threshold value for possible rules with all values of J(A#B) below the threshold being disregarded.

E. From the values of I(A∩B) and J(A#B) that are at or above the threshold, the software should extract the rules (to keep). For the certain rules, the "prime" rules should be extracted. For the possible rules, the "combination" rules should be extracted. For example, if the rules are:

   (1) If tumor is A and B then C is .6.
   (2) If tumor is A then C is .6.
   (3) If tumor is B then C is .6.

   For the certain rules, we extract (2) and (3). For the possible rules, we extract (1).

F. Convert the inclusion {I(A∩B)} and intersection {J(A#B)} symbols to english statements. The purpose of this is to help the user to better distinguish the output.

4.1.3 Output

The software should produce:

A. A complete listing of all rules (certain and possible) in english for :
   1) Before comparison to the threshold value, and
   2) After the comparison to the threshold value.

B. The two threshold values the user has assigned.

C. The final list of extracted certain and possible rules in english.

- SECTION 5 -

CONCLUSION

The decision-making process can easily be simulated by computers if no uncertainty is present. Unfortunately, this is not always the case. Uncertainty may arise due to a number of reasons. It may be due to ambiguity of terms, the skepticism of rules, or by missing or erroneous data. Therefore, methods other than classical logic must be developed to counteract this.

This paper has examined the methods of Dempster-Shafer, Rough sets, and Fuzzy sets in an attempt to achieve the optimal solution. The solution offered here is an integration of these methods. The decision-maker may enter in conditional attributes and decisional attributes. These values are converted into fuzzy set form. From here, lower and upper approximations of belief are established. These approximations generate the rules, certain and possible, which will be used as the basis of the decision-making process.

This entire process is simulated in the program: The *Culas-Worm* Decision-Maker. In this program, the user, or decision-maker, is first given the choices of examining a sample running of the program or

entering in their own data. The data may be entered in fuzzy set form (i.e. inclusive values between 0 and 1) or real form (any real number). From there, the lower and upper approximations and the certain and possible rules are generated. A threshold value may be entered to provide a more "consistent" view of these rules. The rules generated through this process form the foundation for which the user makes their decisions.

This program should satisfy the immediate objective of the research and implementation of the previously referenced methods. In addition, it should provide a foundation for the ultimate long-range goal: the designing of a decision-making process which neutralizes uncertainty.

References

Chang, S.K., Wang, P.P Fuzzy Sets: Theory and Application to Policy Analysis and Information Systems.
Plenum Press, New York (1980).

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.

de Korvin, A., Kleyle,R. *A Unified model for data acquisition and decision making,* Journal of the Amer. Soc. for Info. Science 15 pp. 149 - 161 (1989).

de Korvin, A., Kleyle, R., Lea, R. *An evidential approach to problem solving when a large number of knowledge systems are available,*
The Intern. Journal of Intelligent Systems 5, pp. 293 - 306 (1990).

Dubois, D., Prade, H. Fuzzy Sets and Systems: Theory and Application.
Radius Press, New York (1980).

Ghezzi, C., Jazayeri, M., Mandrioli, D. Fundamentals of Software Engineering. Prentice Hall, New Jersey (1991).

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

Grzymala-Busse,J.W. *Dempster-Shafer theory interpretation of rough set approach to knowledge aquisition under uncertainty,* Tech. Rept.,
University of Kansas, Lawrence,KS (1991).

Kachigan, S.K. Statistical Analysis: An interdisciplinary introduction to Univariate and Multivariate Methods. Radius Press, New York (1982).

Mynatt, B.T. Software Engineering with Student Project Guidance. Prentice Hall, New York (1990).

Pawlak,Z. *Rough Sets: Basic notation*, Institute Comp. Sci., Polish Acad. Sci. Rep No. 431, Warsaw (1981).

Pawlak, Z. *Rough Sets*, Int. Journal Information Computer Sci. 11 pp. 341 - 356 (1982).

Pawlak, Z. *Rough Sets and Fuzzy Sets*, Fuzzy sets and systems 17 pp. 99 - 102 (1985).

Shafer, G. Mathematical Theory of Evidence. Princeton Press, New York (1976).

Strat, T.M. *Decision analysis using belief functions*, International Journal of Approximate Reasoning 4, pp. 391 - 417 (1990).

Yager, R.R. *Decision making under Dempster-Shafer uncertainties*, Tech. Report, Iona College. Report MII-915.

Zadeh, L.A. *The rule of fuzzy logic in the management of uncertainty in expert systems*, Fuzzy sets and systems 11, pp. 119 - 227 (1983).

Appendix A

Program:

The *Culas-Worm* Decision-Maker

```
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;
     condition_2      :  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_array;
   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);
begin                                              { READ SITUATION }
  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;
  writeln('Please enter the two attributes of ',con.consider,' ',
'we will be looking 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 }

{*****************************************************************
****}
{ * * * * * * * * * * * * * * * * * * * * *        R E A D    R E A L
*****************************}
{****
****}
{****  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;
begin                                            { READ REAL }
   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, ': ');
   readln(dec_1);
   write('Please enter a number you associate with definitely
',dec.secattr, ': ');
```

```
   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
.value_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
.value_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
,case_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;
end;                              { READ REAL }


{*****************************************************************
*********}
{***************     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;
```

```
begin                                    { create_cond1_fuzzy_sets }
  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;
end;                                  { create_cond1_fuzzy_sets }


{*****************************************************************
**********}
{******************        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;
begin                                    { CREATE COND_2 FUZZY SETS }
  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;
end;                                     { CREATE COND_2 FUZZY SETS }



{*****************************************************************
**********}
{**************      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;
begin                                    { CREATE DECISION FUZZY SETS
```

```
}
  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;
end;                                    { CREATE DECISION FUZZY SETS
}

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


{*****************************************************************
*******}
{*******************       PROCEDURE    SET    CONDITIONS
************************}
{****
 ****}
{****      This procedure assigns the conditional attributes
 ****}
{*****************************************************************
*******}
procedure set_cond ( con  :  cond_struct;
                     var arr1 : value_array);
begin                                   { SET CONDITION }
   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;
end;                                    { SET CONDITION }


{*************************************************************
*******}
{**********************       PROCEDURE    SET    DECISION_1
**********************}
{****
 ****}
{****    This procedure assigns the first decisional attribute
 ****}
{*************************************************************
*******}
procedure set_dec1(dec : dec_struct;
                   var   arr : value_array);
var
   i  : integer;
begin                                    { SET DECISION_1 }
   for i := 1 to 8 do
     arr[i].dec_kind := dec.firattr;
end;                                     { SET DECISION_1 }


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


{*****************************************************************
********}
{**********************        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;
                      co1att1,co1att2,co2att1 : fuzzy_array;
                      co2att2,decatt : fuzzy_array;
                      var new_arr : value_array);

var
  temp_array  : fuzzy_array;
begin                                             { VALUE INTER }
   new_arr[1].value := num(n,co1att1,decatt);
   new_arr[2].value := num(n,co1att2,decatt);
   new_arr[3].value := num(n,co2att1,decatt);
   new_arr[4].value := num(n,co2att2,decatt);
   inter(n,co1att1,co2att1,temp_array);
   new_arr[5].value := num(n,temp_array,decatt);
   inter(n,co1att1,co2att2,temp_array);
   new_arr[6].value := num(n,temp_array,decatt);
   inter(n,co1att2,co2att1,temp_array);
   new_arr[7].value := num(n,temp_array,decatt);
   inter(n,co1att2,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 ',tval:2:1);
                              w   r   i   t   e   l   n   (   '
------------------------------------------------------------------
------');
                              w   r   i   t   e   l   n   (   '
------------------------------------------------------------------
------');
   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);
                                                { CHECK THRESHOLD }
begin
   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];
begin                                          { PRINT ENGLISH }
   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;
end;                                                { PRINT ENGLISH }


{*****************************************************************
********}
{*****************       PROCEDURE   POSSIBLE   EXTRACT   HEADER
*****************}
{****
  ****}
{****  This procedure generates the heading for extracted rules.
  ****}
{*****************************************************************
********}
procedure possible_extract_head;
    begin                                          { POSSIBLE EXTRACT HEAD
}
        writeln(' ');
        writeln(' ');
        writeln(' These are the extracted possible rules ');
        writeln('--------------------------------------');
        writeln;
    end;                                               { POSSIBLE EXTRACT
RULES }


{*****************************************************************
********}
{*********************       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;
begin                                              { EXTRACT RULES }
   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;
end;                                               { EXTRACT RULES }
```

```
{*************************************************************
*******}
{************************        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 );
   begin                                      { ASSIGN VALUES }
        arr1[1].condition_1.value_att1    :=    0.3;
arr1[1].condition_1.value_att2 := 0.8;
        arr1[1].condition_2.value_att1    :=    0.2;
arr1[1].condition_2.value_att2 := 0.9;
         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 := 0.7;
        arr1[2].condition_2.value_att1    :=    0.4;
arr1[2].condition_2.value_att2 :=0.7;
         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 := 0.4;
        arr1[3].condition_2.value_att1    :=    0.6;
arr1[3].condition_2.value_att2 := 0.7;
         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_att2 := 0.5;
        arr1[4].condition_2.value_att1    :=    0.3;
arr1[4].condition_2.value_att2 := 0.8;
         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 := 0.7;
        arr1[5].condition_2.value_att1    :=    0.2;
arr1[5].condition_2.value_att2 := 0.5;
         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 := 0.2;
        arr1[6].condition_2.value_att1    :=    0.8;
arr1[6].condition_2.value_att2 := 0.2;
         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 := 0.6;
        arr1[7].condition_2.value_att1    :=    0.7;
arr1[7].condition_2.value_att2 := 0.1;
         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;
begin                                              { EXAMPLE }
   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
                     w  r  i  t  e  l  n  (  '
**********************************************');
                     w  r  i  t  e  l  n  (  '
**********************************************');
      writeln('    **
**');
```

```
      writeln('     **  WELCOME TO THE CULAS-WORM DECISION MAKER
**');
      writeln('    **
**');
                      w  r  i  t  e  l  n  (  '
**********************************************');
                      w  r  i  t  e  l  n  (  '
**********************************************');
      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;
  ch : char;
begin                                          { READ FUZZY }
    ch := 'y';
    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;
      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_att1:2:1,
           '   ':10,arr1[i].condition_1.att2:10,'    =
':2,arr1[i].condition_1.value_att2:2:1);
           writeln(arr1[i].condition_2.att1:10,'    =
':2,arr1[i].condition_2.value_att1:2:1,
           '   ':10,arr1[i].condition_2.att2:10,'    =
':2,arr1[i].condition_2.value_att2:2:1);
           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);
   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_at
t2_arr);

create_cond2_fuzzy_sets(cases,no_of_cases,cond2_att1_arr,cond2_at
t2_arr);

create_decision_fuzzy_sets(cases,no_of_cases,dec_att1_arr,dec_att
2_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 }
```