1987

A Toolkit for uncertainty reasoning and representation using fuzzy set theory in PROLOG expert systems

Marcelle M. Bicker

Rochester Institute of Technology
School of Computer Science and Technology

A Toolkit For Uncertainty Reasoning and Representation Using Fuzzy Set Theory in PROLOG Expert Systems

by
Marcelle M. Bicker

A thesis, submitted to
The Faculty of the School of Computer Science and Technology
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Approved by:

Professor Al Biles	date
Professor Jefferey A. Lasky	date
Dr. Peter G. Anderson	date

Title of Thesis

A Toolkit For Uncertainty Reasoning and Representation Using Fuzzy Set Theory in PROLOG Expert Systems

I, Marcelle M. Bicker, prefer to be contacted each time a request for reproduction of this thesis is made. I can be reached at the following address:

875 Chili Coldwater Road

Rochester, NY 14624

Signed: Marcelle M. Bicker

Date: Sept. 29, 1987

Abstract

This thesis examines the issue of uncertainty reasoning and representation in expert systems. Uncertainty and expert systems are defined. The value of uncertainty in expert systems as an approximation of human reasoning is stressed. Five alternative methods of dealing with uncertainty are explored. These include Bayesian probabilities, Mycin confirmation theory, fuzzy set theory, Dempster-Shafer's theory of evidence and a theory of endorsements. A toolkit to apply uncertainty processing in PROLOG expert systems is developed using fuzzy set theory as the basis for uncertainty reasoning and representation. The concepts of fuzzy logic and approximate reasoning are utilized in the implementation. The toolkit is written in C-PROLOG for the PYRAMID UNIX system at the Rochester Institute of Technology.

Keywords

Uncertainty, imprecision, data content uncertainty, data context uncertainty, expert systems, credibility, fuzzy set theory, fuzzy logic, fuzzy thinking, knowledge representation, inferencing methods, approximate reasoning, PROLOG, program toolkits.

Computing Reviews Subject Codes

Primary

I.2.1 Applications and Expert Systems
medicine and science
natural language interfaces

Secondary

I.2.3 Deduction and Theorem Proving
answer/reason extraction
deduction
logic programming

I.5.1 Pattern Recognition Models
fuzzy sets

Table of Contents

List of Figures

Chapter 2

Chapter 3

Chapter 1

An Overview of the Thesis

Expert systems have become the fastest growing application of artificial intelligence today. They have been written for utilization in fields ranging from sophisticated machine diagnosis tools to advanced military and scientific applications. There remains, however, critical issues involving expert systems that must be addressed and resolved. The handling of uncertainty or imprecision in expert systems is one such problem.

Chapter 1 in this thesis document presents a brief description of the problem. It identifies a number of alternatives that have been taken in the search for representations of uncertainty. One alternative, fuzzy set theory, was chosen for the implementation of a PROLOG-based toolkit as part of the thesis. An outline of the remaining thesis chapters and a summary of the proposed work is given.

1.1 Problem Specification

The problem of dealing with uncertainty in expert systems requires definitions of both uncertainty and expert systems. Uncertainty in knowledge is data or information that is unclear, incomplete or just not entirely believable.

In traditional computer applications, uncertain data is not accepted or dealt with; all information is certain and precise. Expert systems are more recent computer programs that address expert problems previously solved only by human experts. It has been found that the knowledge of experts used to solve expert problems is inherently uncertain. Therefore, in order for expert systems to emulate the expert's knowledge, the expert system program must be able to handle the uncertainty. Specifically, the program needs the ability to store uncertain data and to reason with the data. This is one of the major problems facing expert systems developers today in dealing with uncertainty in expert systems.

1.2 Different Approaches to the Problem

Several approaches to the problem of dealing with uncertainty in expert systems have been presented. There are five dominant theories today, that will be reviewed in this thesis document. Traditional Bayesian theory was the first alternative to dealing with uncertainty in data, and it emerged long before expert systems. The confirmation theory, used in the Mycin expert system, was one of the first real attempts to deal with uncertainty in expert systems. Dempster and Shafer's theory of evidence expanded on the earlier approaches to provide a powerful probabilistic method. Fuzzy set theory is a relatively new approach

involving possibilities rather than probabilities. Finally, the theory of endorsements, which is a nonnumerical approach, addresses the problem of why data is uncertain.

1.3 Implementing a Fuzzy Set Theory Alternative

As will be discussed, this thesis describes the implementation of an uncertainty toolkit to be utilized by expert systems to deal with uncertainty in a Prolog environment. Fuzzy set theory was chosen as the uncertainty representation and reasoning mechanism for one primary reason. Alternatives to the solution of the problem that are based on probability theory (Bayesian probabilities, confirmation theory and the theory of evidence) are traditional, well-founded, well-documented approaches. Fuzzy set theory is an emerging approach that is just beginning to challenge the older methodologies. It provides exciting, new concepts to an alternative to the problem and is, therefore, more appealing for the toolkit implementation. Nonnumerical methods have made the most recent contributions to a solution of the problem and are neither well-documented nor tested in applications.

1.4 Outline of the Thesis

Chapter 2 of the thesis document provides definitions of uncertainty and expert systems in much greater detail. The importance of uncertainty in expert systems is stressed. A closer look at each of the five alternatives described is then given followed by a comparison of the approaches. Chapter 3 is an extensive description of the PROLOG uncertainty toolkit designed and implemented using fuzzy set theory for the thesis. Careful attention is given to the representation of uncertainty and the uncertainty reasoning mechanism used by the toolkit. A complete review of the environment in which the toolkit runs completes the third chapter, along with an important interpretation of the toolkit results. The final chapter, Chapter 4, contains an overview of the test cases used in the toolkit implementation, a discussion of the problems encountered, and a final analysis of the results. The thesis concludes with some closing remarks and observations.

Chapter 2

Comparing Several Alternatives to the Problem

The first chapter presented a brief description of the problem of dealing with uncertainty in expert systems. It is the intent of this chapter to further define the problem and to examine a number of alternatives in addressing a solution. First, a definition of uncertainty is given, as well as a discussion of expert systems. It is then shown that since the knowledge of experts is inherently fuzzy or imprecise, there is real value in maintaining this uncertainty in expert systems. The problem is how to represent and utilize the uncertainty in an expert system to maximize the credibility of the system.

A number of approaches to the problem are reviewed here in a chronological order. Bayesian theory was the traditional approach to representing uncertainty in data before the development of expert systems. Mycin's confirmation theory was one of the first attempts to represent uncertainty in expert systems. Fuzzy set theory, developed by Lofti Zadeh, utilized possibility theory, unlike the traditional probability theory. The section discussing fuzzy set theory is the most extensive section, since the uncertainty toolkit implemented in this thesis is based on fuzzy set theory. Dempster and Shafer later expanded the probability theory approach in their theory of evidence. Finally, more recent,

nonnumerical methods of dealing with uncertainty are explored. Specifically, Cohen's theory of endorsement is reviewed. The chapter concludes with a comparison of the alternatives described.

2.1 Defining Uncertainty

In defining uncertainty, two aspects can be considered: uncertainty in the content of data and uncertainty in the context of data. Information that is vague, unclear, fuzzy, imprecise, incomplete or even missing constitutes uncertainty in the content of data. For instance, the statement "Steve is tall" contains uncertainty since the term "tall" is not precise. Such statements containing uncertainty are common in the natural language used by humans. In fact, uncertainty in the content of data is inherent in natural language [Negoita 1985].

Uncertainty in the context of data involves the issues of belief and validity. The statement "The height of Steve is 10'" contains no imprecise data in itself but in the context of the real world, it is not believable. The validity of this proposition is uncertain. Degrees of confidence or belief, which represent a level of data context uncertainty in statements, will also be considered in dealing with uncertainty in this project.

2.2 Expert Systems

An expert system can be defined as a computer program that solves expert problems using expert knowledge [Cohen, P. 1985]. It consists of two main components: a knowledge base and an inference machine or engine. The knowledge base is the representation of the information stored by the expert. The inference engine accesses the knowledge base in providing answers to input queries. The method of reasoning used to solve queries is called the inferencing method. Input and output facilities are also included in an expert system to communicate with the user. Acquisition of knowledge and the ability to explain the results of the system are two other important concepts in expert systems not covered in this thesis [Baldwin 1985].

Typically, an expert system maintains a large knowledge base within a very specific domain; it knows a lot about a little. Experience shows that the data in the knowledge base is subjective in nature. It is often judgemental, experiential, 'rule of the thumb' information acquired from the expert [Cohen, P. 1985]. In addition to uncertain data, it is often true that the expert may have varying degrees of confidence in the information. Therefore, the knowledge in expert systems frequently contains both data content and data context uncertainty.

To conclude this, a description of the individuals

involved in building and using an expert system is given. The 'expert' is the person who has the expert knowledge. This may be a person who has been in the field for an extended period of time. Often, there may be multiple experts involved. The 'knowledge engineer' is the individual who gathers information from the expert(s). The knowledge engineer then works with the 'expert system developer' (or just 'systems developer') in constructing the knowledge base and inferencing scheme from the information obtained from the expert. In addition, the expert systems developer may need to tailor an expert system tool to meet specific requirements. Finally, the 'end user' or 'user' includes all those persons who will actively utilize the expert system to solve expert problems previously handled only by the experts. This allows the expert to pursue other activities and problems without sacrificing the benefits of the expert knowledge from the end users.

2.3 The Value of Uncertainty in Expert Systems

Since the knowledge of experts contains uncertainty, the knowledge engineer and expert systems developer must decide how to deal with that uncertainty when constructing an expert system. Uncertainty can be handled in a number of ways. One common method has been to make assumptions about the real world that eliminate uncertainty. For instance, an assumption that a person is tall if he is over six feet in height

increases the precision of the statement "Steve is tall". Such assumptions handle uncertainty implicitly and may be useful in simple applications.

However, the principle of incompatibility [Gaines 1977] contends that uncertainty cannot be eliminated or overlooked in more complex problems. In fact, as the complexity of a problem increases, the ability to express the problem in precise terms decreases. Any attempt to do so results in a loss of relevancy to the original problem.

Therefore, it may be that precision is not only unattainable, but unnecessary and undesirable [Zadeh 1974]. The challenge for the knowledge engineer is to maintain the delicate balance between precision and relevancy. It is necessary to recognize when a given level of uncertainty or precision is not only valid in expert systems, but also valuable in the reasoning method. Since the goal in designing expert systems is to emulate the decision processes of an expert, the limited processing and storage capacities of the human mind must be taken into account. Given these limited resources, the ability to reason under uncertainty is an asset essential to human thinking. It is best then, for the knowledge engineer to deal with uncertainty explicitly in the construction of an expert system in an attempt to model human reasoning as a closer approximation of reality [Gaines 1977].

2.4 Credibility of Expert Systems

The success of an expert system depends heavily on the reliability of its results. The knowledge representation and the inferencing method used by an expert system provide the basis for this credibility [Cohen, P. 1986]. The method used in dealing with the uncertainty inherent in expert system applications is an important aspect. Both data content and data context uncertainty must be definable in the knowledge representation scheme. The inferencing scheme also provides credibility by dealing with uncertainty using one of many inferencing techniques. For instance, the probability theory utilized by Mycin, the possibilities in fuzzy set theory and the explicit representation of the sources of uncertainty in the thoery of endorsements supply the basis for credibility. In all cases, the inferencing scheme must allow for the combination of rules or evidence, either in series (a Boolean 'and' condition) or in parallel (a Boolean 'or' condition) [Cheng, Kashyap 1986].

2.5 Bayesian Theory of Probability

Bayesian probability theory has been used to represent uncertainty in experiments and propositions long before expert systems evolved. The concepts of the theory are based on probabilities and the work of Thomas Bayes (1702-1761).

A description of probabilities in experiments and propositions is first presented. Bayes' theorem is then introduced as a method of modifying probabilities given new evidence or information. The limitations of such a method in expert system applications is discussed in section 2.5.3.

2.5.1 Probabilities in Dealing with Uncertain Data

Unlike a deterministic mathematical model in which the actual outcome of an experiment is derived explicitly from the conditions of the experiment, a probabilistic model cannot predict the results of an experiment specifically. Probability expresses the chance or frequency that an experiment produces an outcome. Such an approach is useful when it is impossible to obtain accurate measurements or precise information [Meyer 1970].

More specifically, probability theory assigns a value to a proposition or a hypothesis that represents the chance the assertion is true or false. This can be done when the validity of the proposition is uncertain (data context uncertainty). The probability X that a hypothesis H is true is represented by:

$$P(H) = X.$$

The value of X is in the range [0,1], where 0 implies the proposition is absolutely false and 1 implies the proposition is completely true. Values within the range represent a

varying degree of belief in the proposition [Meyer 1970].

For instance, if H is the statement "Steve is over 6' tall", then the probability of H might be 30% or

P("Steve is over 6' tall") = 0.3.

Even though there is no uncertainty in the proposition H itself, there is uncertainty in the belief that H is true which is expressed in the probability.

2.5.2 Bayes' Theorem

Frequently, the probability of an outcome is dependent on a previous result. If new or additional information concerning a fact or event is obtained, the probability of the original proposition may require modification. For example, if the fact that "Steve can reach the ceiling" is received in the knowledge base, then the probability that 'Steve is over 6' tall" should be revised to reflect the new information.

The new probability is termed a 'conditional probability,' which is calculated from the prior probability. The conditional probability is based on an additional condition or fact introduced into the knowledge base.

Conditional probabilities are calculated using a theorem called Bayes' theorem [Meyer 1970]. This requires that the probability of the new information be determined along with the probability of all the conditions in the same sample or

event space. Assume that the proposition H_i ("Steve is over 6' tall") is in a sample space of mutually exclusive, completely exhaustive propostions ($H_1 ... H_n$). This implies that the H_is (i=1..n) do not overlap and that one and only one H_i is always true. The sum of the probabilities of the H_i is always equal to 1. Also assume that a new piece of information or evidence (e.g. "Steve can reach the ceiling") is represented by E. Then Bayes' theorem states:

$$P(H_i|E) = \frac{P(H_i)P(E|H_i)}{P(H_1)P(E|H_1) + \ldots + P(H_n)P(E|H_n)}$$

The notation P(H|E) is read as the probability of H given the evidence E. The probability $P(H_i)$ is called the a priori probability or the probability of the proposition before the new data was acquired. Note that Bayes' theorem requires the a priori probability of all the propostions in the sample space as well [Meyer 1970]. While this theorem provides a robust means of computing revised probabilities, it does so at the expense of requiring a large amount of data in determining all the conditional probabilities in the formula. In the example, let H_1 = "Steve is over 6' tall", and H_2 =" Steve is less than or equal to 6' tall" (mutually exclusive and collectively exhaustive). If the probability that Steve can

touch the ceiling if he is over 6' tall equals 90% and the probability that Steve can touch the ceiling if he is less than or equal to 6' tall equals 20%, then the probability that Steve is over 6' tall, given that Steve can touch the ceiling is:

$$\frac{(.3)(.9)}{(.3)(.9) + (.7)(.2)} = .66 \quad \text{or 66\% true.}$$

2.5.3 Limitations of the Approach

Bayesian probabilities can be useful in simple applications as illustrated in the example in Section 2.5.2. However, as the complexity of the problem increases, Bayes' theorem requires an exponentially large number of conditional and a priori probabilities representing opinions on all events in the sample space [Henrion, Wise 1985]. Not only is this a large volume of data, but these additional probabilities are also often unknown. Although mathematically sound, the requirements of Bayes' theorem have the effect of limiting its usefulness as a method of dealing with uncertainty in large expert system applications.

2.6 Mycin Confirmation Theory

One of the first attempts to overcome the restrictions

of Bayes' theorem in a probabilistically based approach to dealing with uncertainty in an expert system was the Mycin medical diagnosis system. A review of the background of Mycin and its predecessor, Dendral, is discussed in Section 2.6.1. Edward Shortliffe, the developer of Mycin, created a unique representation of data uncertainty, called uncertainty factors. Shortliffe also derived the combinatorial formulas from probability theory used to implement the Boolean 'and' and 'or' operators in combining evidence in the Mycin expert system. Unfortunately, certain theoretical objections and restrictions with the deductive reasoning used in Mycin limited its actual implementation in a clinical environment. Finally, another similiar expert system used in ore deposit exploration, called Prosector, is reviewed.

2.6.1 Dendral and Mycin: The First Expert Systems

In the mid 1960s, work was being done in the Stanford Heuristic Programming Project to identify chemical structures and to generate the resulting molecular diagrams from mass spectrograms. It became obvious to the developers that a brute force combinational explosion of all alternatives of atomic structures was impractical. What was needed was the experienced knowledge of experts to prune down the combinations. What resulted was a system called Dendral which is often regarded as the first expert system (although an

expert system was not the original intent). It demonstrated the concept that knowledge is power by replacing vast brute force searches with expert rules of fragment atomic structures [Cohen, P. 1985].

The lessons learned from Dendral were implemented in the first true rule based system, also written at Stanford in 1974 by Edward Shortliffe. Mycin was a consultation tool used to aid physicians in the identification of an infecting organism and in the selction of antimicrobial therapy in infectious diseases. While Dendral involved enormous volumes of data, the challenge in Mycin was to address the uncertainty inherent in clinical decision making [Buchanan, Shortliffe 1984].

Mycin maintained between 400 and 500 rules derived from experts in the field. It used a backward chaining inference method, which began with the condition of the patient and worked backward to determine the disease (the cause) and an appropiate therapy (an action). To handle the uncertainty in the data, the syntax of Mycin rules was modified to permit a degree of belief, called a confidence factor, to be specified. This was a probabilistic weight applied to both propositions and implications. Mycin then utilized modus ponens style reasoning and probability-based formulas to combine the certainty factors.

While Mycin avoided strict Bayesian probability theory, due in part to the large volumes of data required, the confirmation theory adopted in Mycin as an inferencing scheme

was essentially a probability based method [Buchanan, Shortliffe 1984].

2.6.2 Mycin's Confidence Factors

Any fact or rule in Mycin that contained uncertainty was assigned a confidence factor, which measured the degree of belief or misbelief that the expert had in the knowledge. The confidence factor was a number in the range [-1,1], where -1 implied that the fact or rule was absolutely false (complete misbelief), 0 implied no knowledge of the verity of the clause (no belief or evidence) and 1 assumed the fact or rule was absolutely true (complete belief). Thus if an expert believed a rule was 70% true, a confidence factor of 0.7 was assigned to the rule [Buchanan, Shortliffe 1984].

Once the confidence factors were assessed, Mycin used modus ponens or deductive reasoning in the inference engine. Simply stated, if a proposition p implies a conclusion q, then if p is true, then q must also be true (if p then q). Mycin combines the confidence factor of q with the confidence factor of p by multiplication for the resulting confidence factor of the implication. If the rule contains multiple conditions (if p1 and p2 then q), Mycin takes the minimum confidence factor of the conditions before multiplying it with the confidence factor of q. This type of rule combination has also been termed parallel certainty

inferencing [Cohen, P. 1985]. An example of a Mycin-type rule would be:

Rule: if X and Y
then Z with certainty 0.8.

Facts: X with certainty 0.5.
Y with certainty 0.7.

Conclusions: Z with certainty 0.4
= 0.8 * max(0,min(0.5,0.7)).

Mycin takes the maximum of zero and the minimum confidence factor of the conditions to balance the effects of negative or misbelief confidence factors [Neapolitan 1986]. In this manner, Mycin combines the certainties of the premises joined by the Boolean 'and' operator in series to yield the certainty of the conclusion. The next section considers how Mycin treats the Boolean 'or' operator in the inferencing scheme to combine certainties in parallel.

2.6.3 Combining Evidence in Mycin

In some instances, a single conclusion in an expert system may be produced by multiple, conflicting rules. For instance, if a propostion p is true if q1 or q2 is true and the expert system concludes both q1 and q2, then the inferencing machine must combine the evidence of q1 and q2 to generate a confidence factor for p. Mycin approached this

situation by first making the assumption that the conflicting premises were independent. Using probability theory, Mycin was able to define the combination of evidence (using the Boolean 'or' operator) as follows. Assume two Mycin-type rules and two facts:

Rules: If X
then Z with certainty 0.6.
If Y
then Z with certainty 0.7.

Facts: X with certainty 0.5
Y with certainty 0.7.

The inferencing machine determines the confidence factor of both rules:

Z with certainty 0.3
= 0.6 * max(0,0.5) (CF1)

Z with certainty 0.49
= 0.7 * max(0.0.7) (CF2)

The intermediate results are noted as CF1 and CF2 for clarity. The final confidence factor (CF) is then calculated with one of the following formulas:

$CF = CF1 + CF2(1-|CF2|)$ when CF1 and CF2 are both positive or both negative.

$$CF = \frac{CF1 + CF2}{1 - \min(|CF1|,|CF2|)}$$ when one CF1 or CF2 is positive and the other is negative.

Since both CF1 and CF2 are positive, the example produces:

$$CF = 0.3 + 0.49(1 - 0.49) = 0.55.$$

The first half of the formula (when CF1 and CF2 have the same sign) follows from probability theory given the independence assumption [Cheng, Kashyap 1986]. The formula given for CF1 and CF2 with different signs was derived to produce intuitive results and has no apparent foundation in probability [Neapolitan 1986].

The use of confidence factors in Mycin along with the combinatoric methods used to combine premises in serial and in parallel are described by the confirmation theory of Mycin. The limitations of the theory, specifically the independence restrictions, led to further advancements in uncertainty reasoning. This will be discussed further in the sections dealing with Dempster and Shafer's work.

2.6.4 Drawbacks to Deductive Reasoning

An important issue to address when reviewing Mycin is the advantages and disadvantages of deductive reasoning. While the modus ponens style of inferencing is a powerful technique, well founded in symbolic logic, it has certain drawbacks that have hindered the Mycin system. Deductive reasoning in general has a problem with default reasoning.

For instance, assume an expert system contains the knowledge "If X is a bird, then X can fly". Additional information that a penguin is a bird, violates the first rule since it is known that penguins cannot fly (even though they are classified as birds). To maintain the integrity of the system, the rule is changed to "If X is a bird and X is not a penguin, then X can fly". Unfortunately, it is now impossible to conclude that X can fly if all that is known is that X is a bird (it must also be known that X is not a penguin). This restrictive effect of default reasoning in deductive logic can become crippling to an expert system [Michie 1981].

A second criticism of deductive reasoning is that it is unlike human reasoning except in the most simple situations. The specific content of the domain plays a significant role in whether a human will strictly apply the logical rule. A concrete domain usually results in conformance to deductive reasoning. On the other hand, humans react more often to an abstract domain by making conclusions based on inductive reasoning, choosing a solution that is "most typical" or has the "best fit." The latter is quite different from the deductive reasoning used in Mycin [Smith 1985].

The Mycin expert system was never successfully implemented full scale in the medical diagnosis of infectious diseases. The drawbacks of deductive reasoning were one factor. The interdependence restrictions required by the inferencing scheme were another. Inadequate or incomplete explanation facilities have also been cited [Cohen, P. 1985].

In any case, Mycin was unable to provide the high level of credibility which is critical in any medical application. Regardless, the breakthroughs in expert systems technology and in uncertainty inferencing provided by Mycin are invaluable.

2.6.5 Prospector Expert System

A second early expert system worth noting, is the Prospector expert system written at the Stanford Research Institute in the mid 1970s. It assesed the likelihood of ore deposits contained in specific geographical regions for further exploration. Prospector utilized a network inferencing method to gather support of evidence in rules [Michie 1981]. The knowledge base maintained models of different types of ore deposits. Uncertainty was then reflected by a subjective probability (or the evidence of a symptom) in the range [-5,5]. A value of -5 assumed there was no evidence of a symptom. A zero value stated that the presence or absence of a symptom was unknown. A +5 value implied that it was completely true that a symptom existed [Baldwin 1985].

Given the weighted evidence of the presence or absence of symptoms, Prospector utilized the probabilities to compute the support of the final conclusion. It assumed a high level assertation of the presence of an ore deposit and worked to

verify the conclusion in the network, where the field evidence was stored in leaf nodes of the network [Michie 1981]. While Mycin used probabilities to express degrees of belief, the subjective probabilities in Prospector allowed for the implementation of plausible reasoning. A disadvantage in Prospector was that it was often difficult to obtain the necessary probabilities. At best, the subjective probabilities could only be expressed in linguistic terms such as "very probable" or "highly unlikely".

2.7 Fuzzy Set Theory

At about the same time that Mycin was being developed (mid 1960's), a mathematician named Lofti A. Zadeh began to formulate the ideas of fuzzy set theory. The beginnings of fuzzy set theory in the work of Zadeh are traced in the first section (2.7.1). Later developments including approximate reasoning and the possibility/probability consistency principles are described next. This section also includes a review of recent languages, such as PRUF and FRIL, which incorporate the concepts of fuzzy set theory.

Sections 2.7.3 through 2.7.8 contain a thorough description of fuzzy set theory. Uncertainty is represented as fuzzy subsets in terms of a structure called a possibility distribution. Both data content and data context uncertainty are defined. Underlying the uncertainty representation is a concept of truth which must be understood. Finally, the operations of fuzzy logic used to combine the uncertainty, is outlined. The section concludes with criticisms of fuzzy set theory.

2.7.1 Zadeh, The Founder of Fuzzy Set Theory

Zadeh is often regarded as the founder of fuzzy set theory. His work at the University of California at Berkeley originated in complex systems analysis and decision

processes. In the early 1960s, his goals were to develop general mathematical systems theories to be used in the engineering design of circuits and control theory. It became obvious that optimal control theory was at its peak in the 1960's and that such theories were proving to be too precise for real world situations [Gaines, Shaw 1985].

Frustrations in these areas led Zadeh to formulate fuzzy set theory in 1965. At the same time, four major paradigmatic shifts in systems theory were occurring. First, inadequate systems models forced a shift to realism in modeling. The real world should be modelled accurately without distortion or destruction. The model should fit the real world, not the other way around.

Secondly, optimal control theory was oversensitive to uncertainties; it was too precise. A major shift in thought resulted from the idea of including the uncertainties in the model. The remaining two paradigm shifts involved modeling the integration of human and automated decision processes specific to control theory [Gaines, Shaw 1985]. Zadeh played an important role in promoting the new approach to systems theory. His new goal was to model human reasoning; a model based on fuzzy data and linguistic variables.

The early work of Zadeh was not applied to an industrial setting until the early 1970s. At that time a control system in London was built that learned to control a steam engine [Zadeh 1974]. The results of the system were encouraging but indicative of the challenge ahead.

2.7.2 Approximate Reasoning and Possibility Vs. Probability

In the early 1970s, Zadeh developed the concepts of approximate reasoning. This described a translation of natural language statements into fuzzy terms, which were combined using fuzzy set theory and retranslated into natural langauge as an approximation of the original statements. The theory was expanded in 1977 with the publication of Zadeh's possibility/probability consistency principle and the creation of possibility distributions [Zadeh 1979]. While theories of probability were already being used in expert systems, the theories of possibility were relatively new. Zadeh described the difference between probabilities versus possibilities as perceiving probability as a degree of belief (likelihood or frequency) versus possibility as a degree of feasability. While some things may be possible, they may not be probable [Zadeh 1979]. Probability represents the uncertainty that occurs from probabilistic behavior of certain physical events. Possibilities, on the other hand, represent uncertainty due to a vagueness in the subjectivity of events as assessed by humans. Theoretical differences are found in the Bayesian probability theory versus possibilities, which are founded in fuzzy set theory.

Around the same time (late 1970s), Zadeh introduced a semantic representation language for natural language call

PRUF (Probabilistic Relational Universal Fuzzy). Zadeh formalized the concept of possibility distributions as a representation of the inherently fuzzy meaning in natural language. PRUF provided a basis for approximate reasoning; the challenge remained to automate the theories [Zadeh 1978].

One such approach emerged recently in a general query language called FRIL (Fuzzy Relational Inference Language), which was developed in 1982 [Baldwin Zhou 1984]. It contains a user query formulation language to input queries, a knowledge base of fuzzy relations (possibility distributions) and an automatic fuzzy reasoning processor. Compared to PRUF, FRIL is the state-of-the-art implementation of the same concepts of fuzzy set theory and the fuzzy inferencing methods of approximate reasoning. FRIL has potential use in many areas of artificial intelligence and in a variety of end applications (e.g. scientific, medical, etc.). The following sections describe the concepts of fuzzy set theory and possibility distributions (mentioned above) in much greater detail.

2.7.3 A Definition of Fuzzy Set Theory

Fuzzy set theory is the study of a class of sets whose membership function is not two valued. Given a domain of values, also called the universe of discourse, one can describe two types of sets of values: crisp sets and fuzzy

sets [Negoita 1985]. In a crisp set, all values in the universe of discourse are either strictly within or strictly outside the set. The boundary is clearly defined, distinct and sharp. On the other hand, fuzzy sets may contain points in the universe of discourse which may not exist strictly inside or outside the fuzzy set. The boundaries of such a set are vague or gradual [Gaines 1977]. Figure 2.1 and Figure 2.2 illustrate the difference between the two set types.

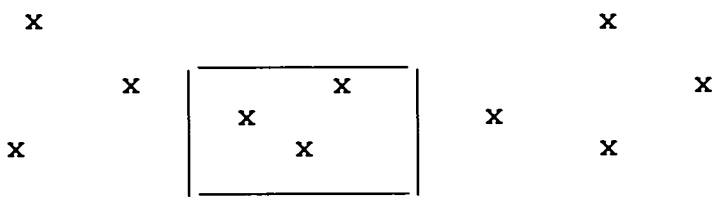

Figure 2.1 Representation of a Crisp Set

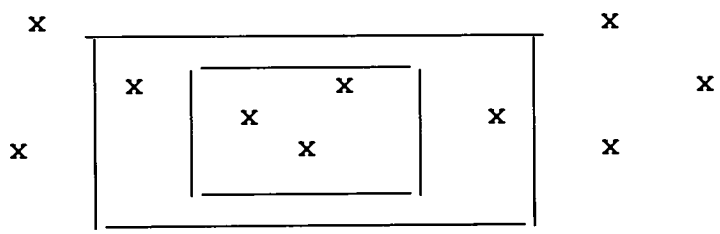

Figure 2.2 Representation of a Fuzzy Set

The 'x's are points that represent values in the set domain. In the crisp set every point is either contained strictly within the set or strictly outside the set. The set of all integer values in the range 0 to 100 is a crisp set. By comparison, a fuzzy set also contains some points entirely within the set or entirely outside the set. However, a fuzzy set contains additional points that are in a fuzzy boundary area of the set and are neither in or out of the set. An example of a fuzzy set is the set of all tall people. Such a

set does not have strict membership reqirements. Where it may be obvious that a person is either very tall (strictly within the set) or not tall at all (strictly outside the set) there are other persons who may or may not be considered tall (contained in the fuzzy boundary area). Fuzzy sets are also refered to as fuzzy subsets. The term fuzzy subset will be used in the remainder of this thesis document.

The fuzziness of a fuzzy subset is determined by the width or expanse of the fuzzy boundary area. This determines the number of points in the domain that lie completely inside the subset, entirely outside the subset, or within the boundary. These categories are represented in fuzzy set theory by a number in the unit interval [0,1] called the degree of membership of a point in the fuzzy subset. The number 1 represents complete membership in the subset and 0 represents complete non-membership [Negoita 1985]. The degree of membership of a point contained within the boundary area can be viewed as the inverse of the distance between the point and the inner boundary edge, assuming that the distance between the boundary edges is no greater that one unit. For instance, if this distance for a point X is equal to 0.6 units, then the degree of membership would be 1 - 0.6 = 0.4. As the distance increases, the degree of membership decreases. Note that the values 1 and 0 are just special cases of this rule for points on or beyond the fuzzy subset boundary edges.

An example is appropiate. Consider the fuzzy subset of

all tall people. The domain is the valid height values for human adults. Assume for simplicity that this domain consists of only six values: 5'2", 5'5", 5'8", 5'11", 6'2", 6'5". The fuzzy subset of all tall people might be described as in Figure 2.3.

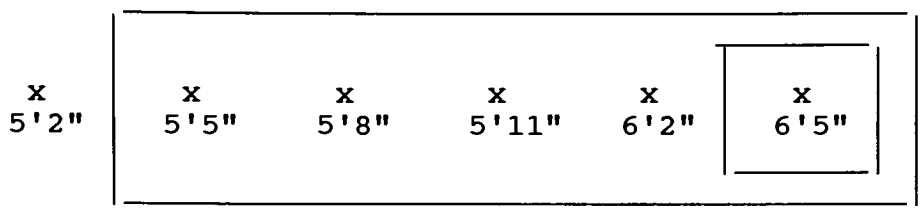

Figure 2.3 Fuzzy Subset of Tall People

Note that persons of height 5'2" would not be in the fuzzy subset at all (degree of membership = 0), while 6'5" people are entirely within the subset (degree of membership = 1). The remaining height values are in the boundary area and have varying degrees of membership. Given such a description of the fuzzy subset of tall people the degrees of membership of these remaining points can be calculated as described above (using the distance function). The entire fuzzy subset can then be represented as a membership function in Figure 2.4.

height	degree of membership
5'2"	0
5'5"	0.2
5'8"	0.4
5'11"	0.6
6'2"	0.8
6'5"	1.0

Figure 2.4 Membership Function of Tall People

Thus if it is known that Steve is 5'11" then his degree of

membership in the fuzzy subset of tall people is 0.6 [Negoita 1985].

2.7.4 Data Content Uncertainty in Fuzzy Propositions

The previous section presented the basic concepts of fuzzy subsets and degrees of membership in fuzzy set theory. The proposition "Steve is 5'11" translated into a specific degree of membership in the fuzzy subset of tall people. However, this statement does not contain uncertainty. Consider the statement "Steve is tall". This proposition contains data content uncertainty. Unfortunately the degree of membership of Steve in the fuzzy subset of tall people becomes fuzzy in itself.

"Tallness" in this proposition is one value of a variable "height" which is an example of a what is called a linguistic variable. Linguistic variables are variables that may have both numerical and linguistic values. For instance, the linguistic variable "height" can have numerical values (5'5", 6'2", etc.) as well as linguistic values (tall, short, etc.). As demonstrated in the previous section, numerical values can have associated degrees of membership values in a fuzzy subset membership function. Linguistic values, on the other hand, have no specific numerical value. Instead, they signify a possibility of a range of values. Linguistic values have been described as labels of fuzzy subsets over a

universe of discourse or a domain of possible numerical values [Negoita 1985]. In the example, this is the six height values. Thus the linguistic value tall, can be represented by the membership function of the fuzzy subset of tall people. This representation is called a possibility distribution [Negoita 1985]. The fuzzy proposition "Ann is short" has a different possibility distribution with degrees of membership in the fuzzy subset of short people over the same domain of numerical height values.

The distinction between numerical values and linguistic values of a variable is clear. Numeric values indicate an actualization of reality, while linguistic values provide only a possibility of a range of values. Data content uncertainty in the latter case is also apparent. It is reflected in the possibility distribution by the increasing degrees of membership as the height numerical values increase. Most people would agree that if Steve is 6'5", he would indeed be tall. If Steve is only 5'11" it is less likely that he would be considered tall (less certain).

2.7.5 Modifying Fuzzy Propositions with Fuzzy Modifiers

Data content uncertainty can be qualified in a statement such as "Steve is tall" by the addition of quantifiers. This has the effect of creating a new fuzzy subset, which in turn can be represented by a new fuzzy proposition. For instance,

in the proposition "Steve is very tall", the term "very" is called a hedge or a fuzzy modifier or quantifier [Negoita 1985]. In terms of fuzzy set theory, the fuzzy subset of all very tall people is slightly different from the original tall fuzzy subset. As seen in Figure 2.5 the new fuzzy subset is smaller, tighter and more concise (less uncertain).

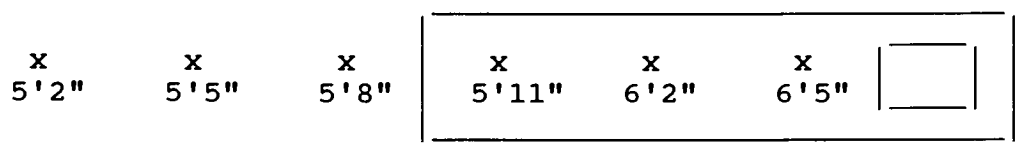

Figure 2.5 Fuzzy Subset of Very Tall People

The fuzzy boundary has both shifted to the higher domain values as well as actually decreasing in size. Two of the domain values have fallen out of the fuzzy subset entirely. The highest value (6'5") once completely within the tall fuzzy subset now resides in the boundary area of the very tall fuzzy subset with a degree of membership equal to the inverse of the distance between the point and the inner boundary edge. The other two points (5'11" and 6'2") are still within the boundary but the distance to the inner edge has increased (decreasing the degree of membership). The resulting membership function is defined in Figure 2.6. This is exactly the possibility distribution of very tall people.

height	degree of membership
5'2"	0
5'5"	0
5'8"	0
5'11"	0.1
6'2"	0.3
6'5"	0.5

Figure 2.6 Membership Function of Very Tall People

In comparison to the membership function of tall people, if Steve is 5'11" it is less likely (lower degree of membership) that he would be considered a very tall person than just a tall person.

2.7.6 Data Context Uncertainty in Fuzzy Set Theory

Data context uncertainty in fuzzy set theory typically decreases the level of uncertainty in a proposition. This has the effect of modifying the boundary area of a fuzzy subset. For instance, if an expert expresses a high degree of certainty in a proposition, "I really believe that Steve is tall," then the boundary area of the fuzzy subset of tall people would diminish. Figure 2.7 describes the resulting fuzzy subset.

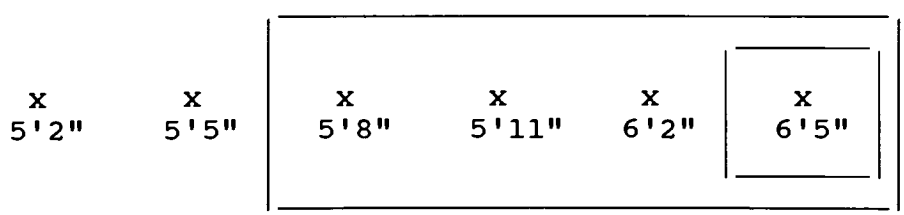

Figure 2.7 Fuzzy Subset of Tall People with Great Confidence

Note that persons of height 5'5" are no longer in the fuzzy subset as the outer boundary of the fuzzy subset shrank. In contrast, the inner boundary has expanded. Domain values 5'8", 5'11" and 6'2" are now closer to the inner edge resulting in larger degrees of membership. The contrasting movement of the boundary edges has reduced the total boundary area, increasing the certainty of the set. Figure 2.8 displays the membership function of the fuzzy subset of tall people given a great degree of confidence in the information.

height	degree of membership
5'2"	0
5'5"	0
5'8"	0.4
5'11"	0.6
6'2"	0.9
6'5"	1.0

Figure 2.8 Membership Function of Tall People with Great Confidence

An expert can express confidence in fuzzy data with the use of fuzzy modifiers. In the example, the term "really" is a fuzzy modifier used to express data context uncertainty. By doing so, the expert is narrowing in on a more distinct subset. The margin of variability becomes smaller and the uncertainty decreases.

2.7.7 Concepts of Truth

Data content uncertainty is inherent in the linguistic

values of linguistic variables. These are represented by the membership functions of fuzzy subsets in the form of possibility distributions. Fuzzy modifiers affect the possibility distributions by restricting the degrees of membership values to a smaller domain set (decreasing the fuzzy subset). Data context uncertainty also modifies possibility distributions by decreasing the uncertainty of linguistic values (narrowing the boundary area). Fuzzy modifiers are used to both modify data content uncertainty and to express data context uncertainty in expert systems. Therefore, fuzzy modifier terms such as 'very', 'really' and 'usually' must be described. Fuzzy set theory maintains an underlying concept of truth upon which fuzzy modifiers are defined [Negoita 1985].

The variable truth can be thought of as a linguistic variable. Assume that truth has numerical values in the unit interval [0,1]. These may be scores on a perfect lie detector test, if you will. Truth also has linguistic values: true and false. Assuming an even distribution of true and false over the numerical values, the possibility distribution of these two linguistic values can be defined. In ascending order within the unit interval, the degrees of membership of true will increase, while the degrees of membership of false will decrease. Given this baseline, the fuzzy modifiers can be determined as modified possibility distributions of true and false in such a way that the resulting possibility distributions are meaningful relative to the original values

of true and false. This will be presented in further detail in the implementation sections of Chapter 3.

2.7.8 Fuzzy Logic

Approximate reasoning allows for the translation of natural language statements into possibility distributions. It also provides a method of inferencing that manipulates the possibility distributions to create others of the same. The new possibility distributions can be retranslated into natural language statements as an approximation of results derived from the original statements. The reasoning method applied in combining the possibility distributions of fuzzy subsets is called fuzzy logic [Negoita 1985].

Fuzzy logic is that logic applied to fuzzy subsets and can be viewed as relations between fuzzy subsets. Unlike traditional two-valued logic, fuzzy logic must deal with ranges of values in possibility distributions. Not unlike traditional logic however, fuzzy logic uses the operations of union and intersection to allow for the conjunction and disjunction of fuzzy subsets respectively. This will become essential in combining expert system propositions in series (using the "and" operator) and in parallel (using the "or" operator). Fuzzy set theory defines these operations on fuzzy

subsets, represented by possibility distributions, as follows:

1) an intersection operation to combine goals defined in conjunction with the "and" operator.

$$f_{R1 \text{ and } R2}(t) = \min(f_{R1}(t), f_{R2}(t)).$$

2) a union operation to combine goals defined in disjunction with the "or" operator.

$$f_{R1 \text{ or } R2}(t) = \max(f_{R1}(t), f_{R2}(t)).$$

3) an inverse function to negate a goal

$$f_{\text{not } R1}(t) = 1 - f_{R1}(t).$$

In all cases, f_{R1} and f_{R2} are membership functions (possibility distributions) on a domain t (the universe of discourse) [Negoita 1985].

As described, fuzzy logic becomes a powerful inferencing tool in reasoning under uncertainty in expert systems. Fuzzy thinking allows for the solution of problems too complex for precision [Zadeh 1974].

2.7.9 Criticisms of Fuzzy Set Theory

One criticism of the use of fuzzy set theory in expert systems is that fuzzy set theory gives no partial credit. All linguistic values in an expert system must be defined as a

fuzzy subset in terms of a possibility distribution. If it is not, then the inferencing scheme will essentially ignore it [Bacon 1986].

A second criticism is that membership functions are often difficult, if not impossible, to define. This is particularly true when the linguistic variable cannot be easily measured. In these cases, numerical values of the domain may not exist and the domain values must be chosen at random. For linguistic values such as "sad" (with linguistic variable equal to "emotion"), the resulting degrees of membership could be questionable.

Finally, the lack of an empirical interpretation and verification of fuzzy set theory is considered a weakness. The statement by Zadeh, that the theory "seems to work" is unacceptable to a more rigorous examination of the approach [Lemmer 1985].

Criticisms aside, the work of Zadeh on the use of fuzzy set theory as a method of dealing with uncertainty in expert systems has provided a fresh approach in viewing the problem. A major impact is in recognizing the differences between possibilities and probabilities. Given that probability theory has been in existence for at least 200 years, the potential for the development of possibilities, as described by the very recent fuzzy set theory, is great.

2.8 Dempster-Shafer Theory of Evidence

Previous sections have discussed the drawbacks of both the Bayesian probability theory and Mycin's confirmation theory. Dempster-Shafer's theory of evidence has emerged as a method of uncertainty reasoning that addresses many of the problems in these earlier systems. This will be discussed throughout the following sections and particularly in the last section which reviews the advantages and disadvantages of the Dempster-Shafer theory.

Arthur Dempster's work in 1967 in belief functions and in upper and lower probabilities was later expanded on by Glenn Shafer in 1976 into a mathematical theory of evidence. Section 2.8.1 explains the basic representation of uncertainty in support of evidence used by Dempster and Shafer. The next section then explores the belief function, which assesses the belief in the evidential support. Dempster's Rule of Combination is discussed in combining evidence that supports conflicting hypotheses in the theory of evidence.

2.8.1 Evidential Support

Dempster-Shafer's theory of evidence is based on the support of evidence rather than on Bayesian probabilities [Gordon, Shortliffe 1984]. Assume that there exists a

proposition in a domain of mutually exclusive, completely exhaustive events (or propostions). Then Dempster-Shafer's theory assigns a degree of support to a set of propositions in the domain rather than a probability to a specific event. The domain is called a frame of discernment in Dempster-Shafer's theory and the support given to a set of propositions in the domain is termed the basic probability assignment or an 'm-value'. If a frame of discernment contains three hypotheses, H1, H2 and H3, then support can be given to any subset of the power set of these hypotheses. For instance, the support (m-value) given to a subset containing H1 and H2 could be expressed as follows:

m(H1 or H2) = 0.5

There exists evidence that supports either H1 or H2 fifty percent of the time. The m-value is assigned in the range [0,1] with 0 representing no support and 1 indicating complete support. The sum of the support (m-values) assigned to the total number of subsets within the frame discernment must equal 1 [Chong, Fung 1985].

Assume that there is evidence that supports H1 70% of the time, H2 20% of the time and there is no evidence in support of H3. Then the remaining 10% can be assigned to a subset containing all three events since it is known that one of the events must be true. The support is described as:

m(H1) = 0.7

m(H2) = 0.2

m(H1 or H2 or H3) = 0.1

All other subsets of the power subset of H1, H2 and H3 have no (zero) support. This method of assigning support to the entire frame of discernment has been called 'discounting' and allows for the assignment of reliability to the results (in this case, the results are 90% reliable) [Cohen, M. 1985].

The assignment of support to subsets of events provides a method of indicating only the support of evidence that is known and relevant to the problem. This is a major advantage over Bayesian theory, which demands an assessment of the probabilities of all events in the domain, as well as a priori probabilities of the events (a much greater task).

2.8.2 Belief Functions and Dempster's Rule of Combination

Once the support of the subsets of the hypotheses are assigned, Dempster-Shafer's theory represents the belief in a set of hypothesis (A) by a belief function. This is calculated as the sum of the m-values of all the subsets (B_i) ($i = 1...n$) of hypotheses within A:

$$Bel(A) = m(B_1) + \ldots + m(B_n)$$

The belief function is also called the lower probability of the subset. A plausibility function exists, which represents the upper probability of A and is determined from the belief function:

$$Pl(A) = 1 - Bel(A')$$

where A' is the complement of A [Chong, Fung 1985].

The upper and lower probabilities described are based in a generalized probability theory. In fact, it can be shown that when support is given exclusively to single hypotheses, the Bayesian probabilities are simply special cases of belief functions [Cheng, Kashyap 1986].

The belief function used in the theory of evidence is a measure of the belief that is held for the set of hypotheses (A). In contrast to a probability, which represents the chance that a hypothesis given the evidence is true, a belief function describes the chance that the evidence means that the hypothesis is true [Cohen, M. 1985]. A subtle shift in focus is made from the truth of the hypothesis to an interpretation of the evidence.

Given the representation of uncertainty in terms of evidential support of subsets, the theory of evidence also provides a method for combining support for conflicting evidence. If two seperate pieces of independent evidence result in conflicting sets of hypotheses, the result must be combined to determine the actual evidential support. For instance, if evidence E1 supports B1 = (H1 or H2) and evidence E2 supports B2 = (H2 or H3), then a subset A, which represents the result of combining B1 and B2, is the intersection of the supported subsets. In this case the common element is H2 or A = (H2). The probability or belief in such a combination is described by Dempster's Rule. Essentially, the probability of any combination of subsets

(across evidence support) is the product of the probabilities of each subset. The probability of any one subset equals the sum of the products of all combinations containing the subset as the intersection. To exclude all combinations that have a null intersection, the result is normalized by a value K [Cohen, M. 1985]. Dempster's Rule is stated as [Chong, Fung 1985]:

$$m(A) = K * \sum_{B_1 \cap B_2} m_1(B_1) * m_2(B_2)$$

For example, suppose an application has three results H1, H2 or H3. The first expert in the field assigns support of 0.95 to H1, 0.5 to H2 and zero support to H3. The second expert, who is equally reliable assigns 0.2 to H1, 0.75 to H2 and 0.05 to H3. The belief functions are defined in Figure 2.9 with the resulting values after applying Dempster's Rule.

	$m_1(x)$	$m_2(x)$	$m_{12}(x)$
H1	.95	.20	.84
H2	.05	.75	.16
H3	0	.05	0

Figure 2.9 Example Belief Functions

Since the experts only assigned support to subsets containing a single hypothesis, the only combinations of subsets with an intersection equal to the subset are $m_1(H1) \cap m_2(H2)$, etc. Assuming K=4.5 (chosen to normalize the results such that the results sum to one), then the belief function of H1 is calculated as

$$4.5 * (.95 * .20) = .84$$

The remaining belief functions are calculated likewise. The results are intuitive. Since the first expert supported H1 almost exclusively, this outweighed the second expert's fairly weak support of H1. The second expert spread out the support across the three hypotheses. The results did reflect the second expert's reluctance of H1 to some extent.

2.8.3 Comparing the Theory of Evidence

There are three main advantages to the Dempster-Shafer theory of evidence over both Bayesian probability theory and Mycin's confirmation theory. The first involves the quality of the evidence. Unlike Bayesian probabilities, which assess the chance that a hypothesis is true, Dempster-Shafer's evidential support supplies the chance that the evidence means (or proves) the truth of the hypothesis. The latter is a stronger statement of the certainty of an event. The quality of the evidence can also be addressed by a simple assessment of the reliability of the evidence in discounting.

The second advantage of Dempster-Shafer's approach is that it does not require such strict, definite numerical valuations of certainty for every hypothesis of the domain. The knowledge engineer can capture only what is relevant, without exceeding the knowledge of the expert or the knowledge that the evidence supports. This is closely related to the third advantage of the theory of evidence: the ability

of the expert to apply support to subsets of evidence, rather than limiting the support to individual events. The additional segmentation of reasoning provides a closer approximation of the expert's knowledge [Cohen, M. 1985]. In addition to these three advantages, the independence assumptions of Dempster-Shafer's theory based in probability theory are less restrictive then the assumptions used in Mycin's confirmation theory [Cheng, Kashyap 1986]. As a generalized probability theory, the theory of evidence is viewed as solving many of the problems of the original Mycin system [Buchanan, Shortliffe 1984].

One of the drawbacks of Dempster-Shafer's approach is in some of the special cases of evidence combination that may produce counter-intuitive results. In the example presented in section 2.8.2, assume the first expert assigned .99 support to H1 and only .01 to H2, while the second expert assigned .99 to H3 and the same .01 to H2. Using Dempster's rule of combination, it can be shown that the result provides complete support to H2 and no support to either H1 or H3. This implies that the reasoning mechanism will absolutely conclude H2, a result that neither expert had any real confidence in. It can be argued that the reliability of the experts should be included by discounting a certain portion of the support to the entire frame of discernment (especially since the experts were in such total disagreement). However, slight variations in the discount used can have a dramatic effect on the results. Therefore, the discount percentage

needs to be chosen carefully and, according to Shafer, should be made independent of any knowledge of the support provided by the experts to the hypotheses. The latter assumption can be restrictive and may be unrealistic in a real world environment [Cohen, M. 1985].

2.9 Nonnumerical Methods

Some of the most recent methods of dealing with uncertainty in expert systems are based on a nonnumerical approach. Certain disadvantages of numerical methods, discussed in the first section, have led researchers to pursue other alternatives. Two of these nonnumerical methods of dealing with uncertainty in expert systems are described. Doyle's reason maintenance is outlined in section 2.9.1. The theory of endorsements presented by Paul Cohen is then discussed, with some final thoughts concerning these nonnumerical approaches.

2.9.1 Drawbacks of Numerical Approaches

It has been argued that numerical methods suffer from a number of drawbacks. While many of the numerical methods are based on probability theory, the events to which degrees of belief are assigned are often not probabilistic in

nature. Therefore, the interpretation of the degrees of belief and the probability rules used to combine them are suspect. Of greater concern is the feedback from experts that the assignment of numbers to express uncertainty is difficult and uncomfortable. This is particularly true when the inferencing method is sensitive to small variations in the numbers used; the expert is then even more uneasy about committing his knowledge to a specific number. In essence, the claim is that numbers are meaningless in representing uncertainty. A range of numbers, as in fuzzy set theory, may be an improvement over a single number, but the numbers still only express how much is believed, rather than why something is believed. Instead of masking the uncertainty implicitly in a numerical value, it is preferable to explicitly represent the sources of uncertainty without numbers [Cohen, Sullivan 1984].

2.9.2 Reason Maintenance

John Doyle developed the concept of reason maintenance in the late 1970s. The uncertainty in an expert system is expressed explicitly as support of evidence in rules (not unlike Dempster-Shafer's theory of evidence). All propositions in the knowledge base are assumed true unless evidence proving otherwise is uncovered. In this manner, assumptions or conclusions in the inferencing scheme depend only on a lack of support. The credibility of the system

relies on the consistency of the rules and on the ability of the system to recover quickly when a contradiction occurs [Cohen, P. 1986].

An example used is one in which the expert system is asked to schedule a room for a meeting, preferably at 10:00 AM in room 813. Reason maintenance inferencing assumes that the meeting will be held at 10:00 AM in room 813. An "in" list and an "out" list of favorable and unfavorable assumptions respectively, is determined to track the network nodes traversed in the inferencing process. At this point the nodes "at 10:00 AM" and "in room 813" are on the "in" list, while "not at 10:00 AM" and "in room 810" are on the "out" list. Additional information is obtained stating that donuts and coffee are allowed in room 813 (on the "in" list) but a projector is not available for that room. Assuming that a projector is required, evidence now exists that does not support the original assumptions. The "in" and "out" lists are modified such that room 813 is now "out" and room 810 is now "in". A projector is available in room 810; therefore the expert system schedules the meeting in this room.

Reason maintenance is one approach to dealing with uncertainty using a nonnumerical method. The problems associated with it focus around the complexity of the expert system. For simple expert systems, reason maintenance may prove effective for missing or incomplete information (one type of uncertainty). However, as the expert system becomes more complex, it becomes more difficult to address every

scenerio in the inferencing scheme. A second problem is in recovering from a proposition assumed true early in the inferencing cycle, which is later contradicted. Both of these restrictions affect response time and limit the usefulness of reason maintenance in large expert system applications.

2.9.3 Theory of Endorsement

Dr. Paul Cohen, of the University of Massachusetts at Amherst, introduced a second nonnumerical concept, called the theory of endorsements, in the search for a method of uncertainty reasoning in expert systems. In 1983, endorsements were presented as an explicit, knowledge-intensive representation of uncertain information. An endorsement is a record of the introduction of uncertainty in reasoning. Endorsements provide reasons to either believe or disbelieve (or both) a proposition which is not completely certain [Cohen, Sullivan 1984]. A positive endorsement reaffirms or strengthens the proposition in the inferencing scheme. On the other hand, a negative endorsement (reflecting disbelief) weakens the proposition. An example of a negative endorsement is the admission of an expert that a premise or a conclusion may be a mistake. Each endorsement in the knowledge base contains a description of the source of the uncertainty in addition to its connotation (positive or negative) [Cohen, P. 1986].

Cohen utilized a plan recognition application program called HMMM to demonstrate the theory of endorsements. If plan1 has three steps (a, b and c) then given step a, the following endorsements are applicable:

(step a is the only grammatical possibility +)

(step a could be a mistake -)

Since plan1 is the only plan available, the only possible step that could be taken is step a (a positive endorsement). However, it may be that step a was a mistake and in fact no plan was desired (a negative endorsement).

Once the endorsements are identified, rules to combine and rank the endorsements are necessary. Typically the combination of endorsements have the effect of erasing negative endorsements to strengthen the belief in the conclusion. For instance, each new consecutive step in plan1 has the effect of weakening the negative endorsement that the first step was a mistake. A general approach to ranking endorsements was applied in HMMM by classifying endorsements as "likely", "unlikely" or "neutral". For example, a negative endorsement is considered "unlikely" and ranked lower in the scheme.

The theory of endorsements provides a unique way of dealing with uncertainty. However, defining a set of endorsements on a knowledge base, along with the combining and ranking rules is a difficult task, as there are no absolutely correct or complete set of endorsements. This subjectivity is a drawback in one sense due to the large

number of endorsements required in a complex expert system and the vast set of combination rules to process the endorsements [Mostow 1985]. On the other hand, if the intent is to represent the major sources of uncertainty, the theory of endorsements is an asset in acquiring and expressing uncertain data from the expert and in the powerful inferencing scheme it projects [Cohen, Sullivan 1984].

Numerical methods of uncertainty reasoning have the advantage of being well founded in mathematical theories of probability and possibility. Uncertainty is represented implicitly in a single number or a range of numbers and is combined using mathematical formulations. Nonnumerical methods rely more on hueristics described by the expert in the form of support or endorsements explicitly expressing the source of the uncertainty. Belief is combined by predefined rules of combination and ranking. Both types of approaches have been implemented successfully, at least to some degree. The argument continues over which should dominate in expert system applications.

2.10 Comparisons and Conclusions

Traditional Bayesian probabilities measure the chance that an event is true. Bayes' theorem is used to calculate the probability that the event or hypothesis is true given new evidence. Unfortunately this approach has been proven ineffective in expert systems due to the large volume of data required (conditional and a priori probabilities for the entire sample space). Although the basic concepts are mathematically sound, Bayesian probability theory is an impractical solution to the problem for anything other than the most simplistic applications.

Mycin's confirmation theory offered the first real breakthrough in dealing with uncertainty. While it is also based in probability theory, it does not require the rigorous definitions of probabilities that the purely Bayesian method enforced. Instead, confidence factors are applied to rules and facts in an expert system to express the degree of belief or misbelief in the knowledge. Probabilistic and hueristic formulas are used to combine the confidence factors in the confirmation theory utilized by the Mycin system.

Mycin suffered a number of drawbacks. Particularly, the pitfalls of deductive reasoning and the restrictive independence assumptions made by Mycin limited the system's usefulness. Other criticisms of confidence factors have been made. For instance, confidence factors are viewed as being too sharp (not fuzzy). Even though rules in the knowledge

base may be probabilistic in nature, the probabilities used to express them are often uncertain themselves. This is not reflected by the crisp confidence factors, reducing the reliability of the expert system. [Zadeh 1984]. Finally, proponents of nonnumerical methods argue that confidence factors are meaningless and are difficult to extract from experts.

Dempster-Shafer's theory of evidence resolved many of the problems of the earlier Bayesian probability theory and the confirmation theory of Mycin. Instead of confidence factors, Dempster and Shafer utilize support of evidence as the basis for uncertainty representation. The belief function measures the probability that the evidence supports the hypothesis, which is a stronger statement of certainty than either of the earlier methods could make. Combining evidence using Dempster's Rule of Combination is still based in probability theory. However, it lacks the narrow independence assumptions that restricted Mycin.

The real advantage of Dempster-Shafer has been the ability to express only the support of evidence that is known and relevant. This includes the opportunity to assign support to groups of hypotheses rather than being limited to individual events.

Critics of the theory of evidence maintain that the method can become too sensitive to the support applied to hypotheses and in special cases, can produce counter-intuitive results. Others maintain that the support used by

Dempster-Shafer suffers from the same disadvantages as confidence factors did in Mycin. The theory of evidence assumes that evidence is crisp; this may be unrealistic in certain applications (where the support itself is uncertain). Likewise, the numbers used to indicate support are essentially meaningless. Many experts may find it quite uncomfortable to express uncertainty in terms of numerical support values. None the less, Dempster-Shafer's theory of evidence has become a dominant methodology in dealing with uncertainty among those who support a probabilistic approach.

Fuzzy set theory was the first mathematical approach to uncertainty reasoning that has been developed independent of probability theory. Uncertainty in fuzzy set theory is represented as a membership function of a fuzzy subset. The reasoning mechanism utilizes the operations of fuzzy logic to combine uncertainty in propositions. Instead of single numerical values used to indicate a probability, fuzzy set theory uses a range of values or a function to represent a possibility. Zadeh maintains that most uncertainty is due to fuzziness rather than randomness and can best be described in fuzzy relations [Zadeh 1984]. Where Mycin measured degrees of belief, fuzzy set theory measures the degree of feasability; some events may be possible but not probable.

Fuzzy set theory has received wide support from those who were dissatisfied with probabilistic methods. Many expert systems that utilize fuzzy set theory techniques to handle uncertainty have been developed, particularly abroad. On the

other hand, proponents of Dempster-Shafer claim that fuzzy set theory lacks a rigorous empirical interpretation, which is a strong point of probability theory. Certainly, the latter has had a longer background than the recent fuzzy set theory, which has only begun to develop. A second criticism of fuzzy set theory is that membership functions are often difficult, if not impossible, to define in certain situations. Finally, the contention that numerical approaches lack meaningfulness has been applied to fuzzy set theory as well. While a range of numbers may be an improvement over a single value, the numbers still do not have any real meaning.

The final approach to dealing with uncertainty has been in nonnumerical methods. Cohen's theory of endorsement is recognized as the leading contender in this category. The concepts of positive and negative endorsements to propositions are presented as explicit methods of expressing the reasons for uncertainty in expert systems. Predefined rules of combination and ranking are applied to manipulate endorsements in the inferencing scheme. The theory of endorsements is a hueristic approach and certainly lacks in any mathematical foundation (disturbing to many, yet appealing to others). As an approximation of human reasoning, one could argue that it is the best approach since humans do not reason using numerical degrees of belief or fuzzy membership functions. However, the theory of endorsements suffers from the same drawbacks as many of the other approaches. Like Bayesian theory, it is a difficult task to

define the complete set of endorsements (or probabilities in Bayesian theory), which can become very large quickly. Selecting the correct endorsements and the proper combining and ranking rules is a challenge. If done properly however, the theory of endorsements can provide a powerful inferencing scheme for many smaller applications.

In conclusion, it is clear that no one method is an obvious best choice. Each method has certain benefits over the others in specific situations. Recall the definition of uncertainty in terms of data context and data content uncertainty. Probabilistic methods (Bayesian theory, confirmation theory and the theory of evidence) all dominate in data context uncertainty. They measure how much is believed or how much the evidence supports the hypothesis. Membership functions in fuzzy set theory, on the other hand, are much stronger representations of data content uncertainty, expressing fuzziness of data rather than randomness. Nonnumerical approaches do not really fall in either category. Instead, these methods represent why knowledge is uncertain. A potential danger in this type of approach is that the precision demanded when one continually questions why knowledge is uncertain may lead to irrelevant information. Zadeh's observation of the inverse relationship between complexity and precision is well taken. Regardless, if the goal in uncertainty reasoning is to approximate human thinking, then it may be that a combination of approaches could provide the flexibility that is required. The unlimited

ability of the human mind to reason under uncertainty is a challenge to be met by all those working in the field.

Chapter 3

Implementation of a Fuzzy Set Theory Solution: A Fuzzy Set Theory Uncertainty Toolkit

Fuzzy set theory can be applied to the problem of handling uncertainty in expert systems as outlined in the previous chapter. The following sections describe an implementation of these concepts in the form of a toolkit designed for expert systems written in the PROLOG programming language. The intent is for expert systems developers to use the toolkit as a way of dealing with uncertainty inherent in expert system applications.

This chapter describes the two main components of the toolkit in detail: the uncertainty representation and the inferencing method. Within the representation scheme, the two types of uncertainty, data content and data context uncertainty, are taken into consideration. The inferencing method includes a powerful thresholding mechanism, which can be used to execute the expert system with varying levels of tolerance to uncertainty. The third section of the chapter describes the exact toolkit environment, including a specific example and an interpretion of the results. Along with the Appendices, this chapter provides sufficient information for an expert system developer to understand and execute an expert system using the fuzzy set theory

uncertainty toolkit.

3.1 Representation of Uncertainty; The First Step

The first step in describing the uncertainty toolkit is to describe the representation of uncertainty in the PROLOG knowledge base. A review of the standard data representation in PROLOG is given to familiarize the reader with the programming language. The toolkit representation of data content uncertainty in the form of fuzzy set theory possibility distributions is described. The concept of truth and a discussion of predefined fuzzy modifiers follows. Section 3.1.4 outlines the method used to qualify data content uncertainty with fuzzy modifiers and introduces a new PROLOG operator. Data content uncertainty can also be applied to the goals in PROLOG rules. Finally, data context uncertainty is examined and a second new PROLOG operator is introduced to specify degrees of confidence in both PROLOG facts and rules. These ideas are described in more detail in the following sections.

3.1.1 Representing Objects and Relationships in PROLOG

PROLOG is a logic programming language that has proven to be useful in constructing expert systems. The knowledge

that PROLOG represents concerns objects (called facts) and the relationship between the objects (called rules). For instance, the fact that Steve is tall is represented as a fact in PROLOG in the form:

```
tall(steve).
```

The rule "a person is big if the person is tall and heavy" is described in PROLOG as

```
big(X) :-
   tall(X),
   heavy(X).
```

The "X" represents an uninstantiated variable, which is replaced by actual values in the inferencing method (discussed later). This is true for any variable beginning with a capital letter. Thus the lower case "s" in the fact tall(steve) means that steve is an instantiated variable (constant).

PROLOG does not represent uncertainty in knowledge. It assumes that every fact and rule is absolutely true or certain. For the types of problems solved by experts in some fields, this assumption cannot be made without sacrificing the validity of both the data and the method of reasoning. Therefore, it is necessary to enhance or expand the knowledge representation in PROLOG to account for both data content and data context uncertainty.

3.1.2 Data Content Uncertainty as Possibility Distributions

As discussed in Chapter 2, data content uncertainty is contained in terms called linguistic variables, which have linguistic values. Consider the statement "Steve is tall". The linguistic variable is "height"; the linguistic value is "tall". Such a linguistic value is described in terms of numerical values contained within a domain and represented in fuzzy set theory as structures called possibility distributions (recall Figure 2.4). It is not important for the PROLOG knowledge base to maintain the linguistic variable of height. In representing data content uncertainty it is important to store the possibility distribution of the linguistic value.

To utilize the toolkit on a PROLOG expert system, every linguistic value contained in the expert system program will be predefined as a possibility distribution. A toolkit utility program will assist the user in defining and reviewing possibility distributions. Domain values and the corresponding degrees of membership in the fuzzy subset will be input and stored in possibility distributions using the PROLOG fact:

```
lingval(Value,Modifier,Number,Degree).
```

Value is the name of the linguistic value (e.g. tall). The modifier will be discussed in the next section. However, when defining the possibility distributions, Modifier will take the null value. Number is one numerical value of the

variable in the universe of discourse (the domain). For instance, 5'10", 5'11" are numerical values of the linguistic variable height. For convience, the height values will be represented as 5.10, 5.11, etc. in the lingval facts, for the remainder of the thesis document. The lingval variable Degree is the degree of membership (a number in the range [0,1]) of the Number in the fuzzy subset labeled by the linguistic value. Since there are multiple numerical values in the universe of discourse, there will be multiple occurrences of this fact for one linguistic value. The example would require the possibility distribution described in Figure 3.1.

```
lingval(tall,null,5.2,0.0).
lingval(tall,null,5.5,0.2).
lingval(tall,null,5.8,0.4).
lingval(tall,null,5.11,0.6).
lingval(tall,null,6.2,0.8).
lingval(tall,null,6.5,1.0).
```

Figure 3.1 Possibility Distribution of Tall People

For consistency in arity, six values will be chosen from the universe of discourse as a representation for each linguistic value. Note that if other occurrences of the same fact exist, the linguistic value would not need to be represented a second time. Thus the statement "Sarah is tall" would not generate additional lingval facts.

The most significant difficulty in this task is defining the universe of discourse of variables that have no real numerical values. For instance, the statement "John is sad" contains the linguistic variable "emotion" which, unlike height, is almost impossible to measure. The best approach

may be to select a scale in the range [1,10] and rank degrees of membership to the fuzzy subset of sad people to these values. It is also wise to choose values from a wide range of values in the universe of discourse as was done with the definition of tall people. This will prevent the representation to be weighted towards any one set of values in the domain.

3.1.3 Predefined Fuzzy Modifiers and Truth Values

The previous section described the basic representation of data content uncertainty in the PROLOG expert system toolkit. This type of uncertainty can be further quantified with terms, such as "very", called fuzzy modifiers or quantifiers. As will be discussed later on, data context uncertainty also relies heavily on fuzzy modifiers to express the degree of belief one has in a PROLOG fact or rule. Before proceeding, an understanding of the representation of fuzzy modifiers in the toolkit is necessary.

Defining fuzzy modifiers like "very", "sortof", or "absolutely" requires a baseline. That baseline is the concept of truth. Truth itself is a linguistic variable with linguistic values of "true" and "false". It is, therefore, possible to identify possibility distributions of these values using numerical values of truth chosen from the unit interval [0,1]. The toolkit defines "true" as in Figure 3.2.

```
lingval(true,null,1,0.0).
lingval(true,null,2,0.2).
lingval(true,null,3,0.4).
lingval(true,null,4,0.6).
lingval(true,null,5,0.8).
lingval(true,null,6,1.0).
```

Figure 3.2 Possibility Distribution of True

Given this baseline, modified values of truth can be defined as possibility distributions. For example, the possibility distribution of events that are very true is represented in Figure 3.3.

```
lingval(true,very,1,0.0).
lingval(true,very,2,0.0).
lingval(true,very,3,0.0).
lingval(true,very,4,0.1).
lingval(true,very,5,0.3).
lingval(true,very,6,0.5).
```

Figure 3.3 Possibility Distribution of Very True

Note that the second parameter, Modifier, in the lingval fact has now been specified with the modifier "very".

The degrees of membership in these fuzzy subsets of the numerical values are chosen in a manner such that the modified values of truth have meaning relative to the unmodified values of truth. Defining a list of valid fuzzy modifiers and categorizing them in terms of the degree of modification (little impact vs. large impact) produces a finite number of fuzzy modifier categories. The degrees of membership for each category have been assigned and represented as possibility distributions. See appendix A for the predefined truth tables.

In compiling the list of predefined fuzzy modifiers, it becomes obvious that two types of modifiers exist: those

representing the degree or extent of validity and those representing the frequency of an occurence or condition. For instance, modifiers such as "very", "extremely", "sortof" and "not_quite" might express how tall Steve is:

Steve is very tall.

While terms like "frequently", "sometimes", "often" and "not_usually" could relate to how healthy Steve is (something which might vary with time):

Steve is usually healthy.

Both types of fuzzy modifiers can be used in representing uncertainty in expert systems using the uncertainty toolkit. See appendix B for a complete list of fuzzy modifiers recognized by the uncertainty toolkit.

3.1.4 Modifying PROLOG for Fuzzy Quantifiers

Understanding the representation of fuzzy modifiers in the uncertainty toolkit allows for further discussion of data content uncertainty. In a previous section, the representation of the data content uncertainty in simple statements such as "Steve is tall" was identified. However, not all propositions are quite as simple. Propositions like "Steve is very tall" are very common. The expert systems developer needs a different PROLOG syntax to define these types of facts. A new functor was introduced, the colon character (:), which has as arguments the fuzzy modifier and

the original simple fact. This takes the form:

```
:(tall(steve),very).
```

For readability, the colon was defined as a new operator in PROLOG and written in the form:

```
tall(steve):very.
```

Thus the expert system developer can quantify the linguistic values in the expert system with fuzzy modifiers. Since the fuzzy quantifier modifies the degrees of membership in the fuzzy subset, a new possibility distribution will be required for the linguistic value "very tall". This possibility distribution will be created automatically by the toolkit (using the concept of very true). This is accomplished by combining the possibility distributions of very true (predefined by the toolkit) with the possibility distribution ot tall people. The resulting possibility distribution will be discussed further in later sections.

3.1.5 Fuzzy Modifiers in PROLOG Rules

Until now the discussion has involved representing data content uncertainty in facts. The same can be applied to the goals of PROLOG rules. If there exists a rule "X is beautiful if X is quite tall and X has very curly hair", then it can be represented in PROLOG as:

```
beautiful(X) :-
    tall(X): very,
    curly_hair(X): very.
```

Multiple goals can be specified within a rule using PROLOG conjunction and disjunction syntax modified by fuzzy quantifiers. Since the goals of a rule are not known facts or realities, the toolkit will not handle these in the same manner that facts are represented as possibility distributions. Instead, the goals serve as conditions of satisfying the rule. The fuzzy modifiers have the effect of further qualifying those conditions and will be treated by the toolkit as an additional requirement in satisfying the goals. The inferencing scheme will support a thresholding device that will prescribe the degree to which the possibility distribution of the goal clause (with the modifier) must match the possibility distribution produced in satisfying the goal. Although this will be described in greater detail in later sections, an example is in order here. Given the "beautiful" rule above and the fact:

```
tall(steve).
```

then in processing the question:

```
beautiful(steve).
```

the goal clause:

```
tall(X):very,
```

will only be satisfied if the threshold allows a loose interpretation of "very tall". If the threshold is strict, the goal will fail since all that is known is that Steve is

tall (not that Steve is very tall).

A final note in this section is to address a method of handling sentences such as "X is absolutely mad if X has a red face and X is screaming'. Since degrees of confidence are not allowed on the head of a rule, the following rules, which decompose the intent, are suggested as alternatives:

```
furious(X) :-
   mad(X):absolutely.
mad(X) :-
   red_face(X),
   screaming(X).
```

3.1.6 Defining Data Context Uncertainty in PROLOG

Previous sections have covered data content uncertainty in linguistic variables possibly modified by fuzzy quantifiers. Data context uncertainty is equally important and will be discussed in this section.

Expressing a high degree of belief or confidence in a fuzzy clause has the effect of reducing some of the uncertainty inherent in the proposition. This is true for both PROLOG rules and facts. The syntax of PROLOG was modified to represent this type of uncertainty with a double dash operator:

```
tall(steve)--very.
```

This fact is different from the fact:

```
tall(steve):very.
```

The former states "It is very true that Steve is tall" while the latter represents the statement "Steve is very tall". Confidence in rules is programmed as:

```
handsome(X) :-
    tall(X),
    curly_hair(X)--absolutely.
```

It is absolutely true that if X is tall and X has curly hair, then X is handsome.

As was true with the fuzzy modifiers applied to goals in PROLOG rules to modify data content uncertainty, data context uncertainty is not represented explicitly by the toolkit beyond the modified syntax. Instead, data context uncertainty will have the effect of influencing the degrees of membership of possibility distributions in a way that the certainty of a fact or rule will be updated according to the data context fuzzy modifier. This is handled by the inferencing method and will be discussed in sections to follow.

Data content and data context uncertainty can now both be expressed in single facts and rules:

```
tall(steve):quite--very.
```

It is very true that Steve is quite tall.

```
handsome(X) :-
    tall(X):very,
    curly_hair(X):quite--absolutely.
```

It is absolutely true that if X is very tall and X has quite curly hair, then X is handsome. These syntax enhancements

along with the predefined fuzzy modifiers, allow for a rich vocabulary for describing uncertainty in expert systems.

3.2 An Uncertainty Reasoning Mechanism; The Second Step

The second step in defining the PROLOG uncertainty toolkit is a description of the inferencing method to be used. PROLOG's inferencing method is first outlined in this section. It is then shown that the PROLOG uncertainty toolkit will enhance the basic PROLOG inferencing method in a number of ways. First, the modified syntax of PROLOG required to express fuzzy modifiers has to be dealt with along with the resulting modifications to the underlying possibility distributions. Secondly, the toolkit must play a role in satisfying facts or rule clauses given the indistinct nature of the possibility distributions representing them. A thresholding technique will be applied to vary the level of tolerance used by the expert system in satisfying terms. Finally, clauses within a rule need to be combined using the fuzzy logic operators of intersection (or), union (and) and the 'not' operator. Details of these enhancements follow in the next sections. A description of the use of approximate reasoning in the toolkit concludes this section.

3.2.1 An Explanation of the PROLOG Inferencing Method

PROLOG uses deductive reasoning with backtracking as a method of reasoning. It is initiated by an input query or

question in the form of a clause such as tall(steve). The inferencing mechanism of PROLOG begins by searching through its database for a match on the input clause. Matching a fact will generate immediate success. Rules are matched by first instantiating all possible variables in the head of the rule. The terms of the rule are then processed sequentially until either all goals are satisfied or a goal fails (according to the rules of conjunction and disjunction). Goals are satisfied by instantiating all known variables and searching through the knowledge base for a match (as above). For instance, given the rule:

```
big(X) :-
  tall(X),
  heavy(X).
```

If the input clause is big(steve), the rule is processed by satisfying the goals of the rule. The variable X is instantiated to steve and the goals of the rule are then satisfied with this instantiated variable. If it is known that tall(steve) and heavy(steve), the rule is completely satisfied. If a goal in a rule fails, backtracking occurs such that alternate paths can be pursued. If no rule can be completely satisfied, the result of the query is negative. A positive result is produced if the query can be successfully matched.

3.2.2 Fuzzy Modifiers in Possibility Distributions

Sections 3.1.4 and 3.1.6 introduced two new PROLOG syntax structures involving fuzzy modifiers to alter possibility distributions. The first incident is the use of fuzzy modifers to quantify data content uncertainty. The statement "Steve is very tall" is an example and is written in the PROLOG toolkit as:

```
tall(steve):very.
```

The possibility distributions of both "tall" (Figure 3.1) and "very" (Figure 3.3) have been presented. Within the toolkit the possibility distribution of "tall" will be predefined by the developers of the expert system and the possibility distribution of "very" is supplied along with the toolkit in the truth tables. The toolkit will dynamically create the possibility distribution of "very tall" using the minimum operator (discussed further in section 3.2.5). The degrees of membership of the corresponding domain values will be combined using the minimum function to create the possibility distribution in Figure 3.4 (as represented in the toolkit).

```
lingval(tall, very, 5.2, 0).
lingval(tall, very, 5.5, 0).
lingval(tall, very, 5.8, 0).
lingval(tall, very, 5.11, 0.1).
lingval(tall, very, 6.2, 0.3).
lingval(tall, very, 6.5, 0.5).
```

Figure 3.4 Possibility Distribution of Very Tall People

The second parameter in the lingval fact now takes the fuzzy modifier value of "very". Note that this combining effort is

applied only to PROLOG facts. Data content fuzzy modifiers expressed in rule clauses will only be treated in the thresholding scheme to be described.

Possibility distributions are also altered by fuzzy modifers expressing confidence factors using the double dash operator:

```
tall(steve)--really.
```

The toolkit interprets this statement as "I really believe that Steve is tall". It modifies the tall possibility distribution in the following manner. The toolkit examines the possibility distribution for non-zero degrees of membership. For all such values, the toolkit decreases low values to zero if possible and increases high values proportionally towards complete membership (value of one). The intent is to shift the boundaries of the fuzzy subset such that the total boundary area decreases. Recall Figure 2.3 and Figure 2.7. As the outer boundary shifts right, domain values previously within the boundary are no longer in the fuzzy subset (complete non-membership, degree of membership = 0). As the inner boundary moves left, some points in the boundary area are now completely within the fuzzy subset (degree of membership = 1). The distance between the inner boundary edge and the remaining boundary area points decreases, thus increasing the degrees of membership of these points. Obviously, the rate of change of points closer to the inner edge is greater than those points closest to the outer edge. The resulting increase in degrees of membership

of these boundary area points is adjusted accordingly.

Another consideration to mention is that different fuzzy modifiers have varying effects in decreasing the uncertainty in fuzzy subsets. In the example, Figure 3.5 describes the new fuzzy subset for tall(steve)--really. Figure 3.6 describes the new fuzzy subset for

tall(steve)--absolutely.

which expresses more confidence in the fact than the previous case.

```
lingval(tall, null, 5.2, 0).
lingval(tall, null, 5.5, 0).
lingval(tall, null, 5.8, 0.4).
lingval(tall, null, 5.11, 0.6).
lingval(tall, null, 6.2, 0.9).
lingval(tall, null, 6.5, 1.0).
```

Figure 3.5 Possibility Distribution of Tall People with Great Confidence (Really)

```
lingval(tall, null, 5.2, 0).
lingval(tall, null, 5.5, 0).
lingval(tall, null, 5.8, 0.4).
lingval(tall, null, 5.11, 0.7).
lingval(tall, null, 6.2, 1.0).
lingval(tall, null, 6.5, 1.0).
```

Figure 3.6 Possibility Distribution of Tall People with Extreme Confidence (Absolutely)

The "absolutely" fuzzy modifer effectively reduces the boundary area to a greater extent than the "really" fuzzy modifier. Therefore, confidence in a clause can be given in various degrees.

Data context uncertainty can also be expressed at a rule level. Assuming that every rule has a resulting possibility distribution (to be discovered), then the same approach can

be taken to modify these possibility distributions to express confidence modifiers defined on rules.

3.2.3 Resolving Clauses Based on Possibility Distributions

PROLOG's inferencing scheme handles the resolution of input queries in the same manner as satisfying clauses within a rule. In both cases, the inference method must determine if a matching fact or rule adequately satisfies the clause in question. This is trivial in basic PROLOG. However, the representation of uncertainty in the PROLOG toolkit complicates the situation. Since facts are represented by possibility distributions, these must be taken into account when satisfying an input query or rule clause. For example, if the expert system contains the knowledge:

```
tall(steve):very.
```

then the inferencing method must determine if the input query:

```
tall(steve).
```

can be satisfied. The possibility distributions of both the known fact and the clause to be satisfied must be determined and compared.

In examining possibility distributions, the toolkit favors two conditions: possibility distributions representing small, tight fuzzy subsets, which are weighted towards the high end of the domain and possibility distributions with

little uncertainty (small boundary area). The first condition expresses high data content certainty while the second represents a large degree of confidence or data context certainty. In order to recognize these conditions, the toolkit has incorporated a type of rating scheme that grants points for favorable conditions. The toolkit interrogates each degree of membership value in a possibility distribution. Complete membership or non-membership values are rewarded high rating points since they indicate absolute certainty of a single domain point. Complete non-membership is given higher preference in order to accomodate the effects of data content uncertainty. This is due to the fact that the concise fuzzy subsets created by large degrees of data content uncertainty shift the fuzzy subset to the high end of the domain, excluding the lower domain points from the fuzzy subset altogether. The rating scheme grants three points to a degree of membership value of 0 and two points to a value of 1.

If a degree of membership of a domain point is not equal to 0 or 1, the domain point must be in the boundary area of the fuzzy subset. The rating system uses the actual value of the degree of membership as the rating points. This has the two effects. First, since these values are always less than one, the boundary area domain points will be given smaller ratings than points strictly in or outside the fuzzy subset. The toolkit rewards fuzzy subsets with a small boundary area (less uncertainty). Second, the toolkit grants a higher

rating to fuzzy subsets with a higher degrees of membership values within the boundary area. Such boundary points are closer to the inner boundary edge than those boundary points with lower degrees of membership (given the inverse distance function). The latter would reflect a looser, less certain fuzzy subset, which is not as desirable.

In the example, the toolkit would rate the known fact:

tall(steve):very.

with a 9.9 rating and the input query:

tall(steve).

with a 7.0 rating. Since the knowledge base fact has a higher rating, the toolkit would assume that the input query is satisfied. Intuitively, this makes sense; if it is known that Steve is very tall, then if questioned whether Steve is tall, the answer is obvious. Not only does the expert system know that Steve is tall, it has the knowledge that Steve is very tall (granted a higher rating).

3.2.4 Using Thresholds in Satisfying Goals

The case where a known fact more than satisfies an input query or a rule clause has been reviewed. Now consider the opposite case. Assume that the knowledge base contains the fact:

tall(steve).

and it is queried with the clause:

tall(steve):very.

The ratings remain the same, only the orientation has changed. Now the toolkit must determine if the knowledge that Steve is tall is sufficient to satisfy the query: "Is Steve very tall?".

To assist the toolkit inferencing logic, a technique involving threshold values has been incorporated in the toolkit. As the user begins to run the expert system, the toolkit prompts the user for a threshold value (this can also be overridden with a default threshold). Acceptable values range from 0 to 50, with 50 being the highest, most stringent threshold. Once set, the toolkit can utilize this threshold value in satisfying clauses.

If the toolkit determines that the possibility distribution of a known fact has a lower rating than the possibility distribution of the clause to be satisfied, the toolkit will calculate the magnitude of the difference between the two ratings. This difference is translated to an acceptable threshold value using a threshold mapping function described in Appendix C. This calculated threshold is then compared to the user's threshold. If it is less than the user's threshold, the clause will be satisfied.

If the calculated threshold is higher than the user's threshold, the clause will fail. As the difference between the ratings increases, the acceptable threshold value calculated by the toolkit decreases. Higher thresholds will not permit clauses to be satisfied unless the possibility

distributions are close. In this way the user can execute the expert system under different conditions. Specifying a high threshold will cause the toolkit to run the expert system with little tolerence to differences in uncertainty. A lower threshold causes the toolkit to be more forgiving in satisfying clauses.

This threshold scheme as applied to input queries satisfied by facts is also utilized in satifying goals within a rule. For instance, given the rule:

```
big(X) :-
  tall(X):very,
  heavy(X).
```

and the knowledge:

```
tall(steve).
heavy(steve).
```

then if queried:

```
big(steve).
```

the result will depend on the selected threshold. The toolkit calculates a rating difference between the knowledge that "Steve is tall" and "Steve is very tall" as a value of 2.9 (see section 3.2.3). This translates into an acceptable threshold of 40 (see Appendix C). If the user indicated a higher threshold (e.g. 50), the clause will not be resolved and the rule will fail. A threshold less than 40 will result in the satisfaction of the goal.

Assuming a rule is satisfied within the threshold, the result of the clause will not be directly affected by the

fuzzy modifier ("very" in the example). Instead, the possibility distribution of the resulting clause will be that of the fact or goal satisfying the clause, unaffected by the "very" modifier. This is done to maintain the actual knowledge in the knowledge base. Steve is big if Steve is very tall and Steve is heavy. Given a liberal threshold, the fact that Steve is just tall satisfies the first condition. However, the toolkit will not assume that Steve is very tall. It will utilize the possibility distribution of:

```
tall(steve).
```

as the result of the rule clause:

```
tall(X):very.
```

This result will be combined with the possibility distributions of the other goals in the rule as discussed in the next section.

3.2.5 Combining Rule Clauses Using Fuzzy Logic

Section 2.7.8 included a description of fuzzy logic operators as prescribed to fuzzy subsets by fuzzy set theory. This section and the next discuss the use of the 'and', 'or' and 'not' operators by the toolkit in combining the possibility distributions of a rule.

The 'and' operator is used most heavily in PROLOG expert systems. Virtually every rule relies on the fact that multiple conditions are met. Recall the big(X) rule of

section 3.2.1. A person is big if he/she is tall AND heavy. Fuzzy logic dictates that uncertainty in rule clauses be handled by combining the possibility distributions of the clauses using the minimum operator on corresponding degrees of membership (relative to their orientation within the domain). A fuzzy subset of heavy people is introduced in Figure 3.7.

weight	degree of membership
100	0
130	0.1
160	0.3
190	0.6
220	0.7
250	0.9

Figure 3.7 Membership Function of Heavy People

Combining this with the fuzzy subset of tall people creates a membership function of tall and heavy people in Figure 3.8.

height	weight	degree of membership
5'2"	100	0
5'5"	130	0.1
5'8"	160	0.3
5'11"	190	0.6
6'2"	220	0.7
6'5"	250	0.9

Figure 3.8 Membership Function of Tall and Heavy People

Notice that the domain of the fuzzy subset has expanded to include both height and weight values [Negoita 1985]. If the possibility distributions of the two clauses to be considered were based on the same domain, this phenomena would not occur. The toolkit handles the problem of different domains by essentially choosing not to attempt to maintain all domain values as possibility distributions are combined. The toolkit

is more concerned with the degrees of membership of the domain points than by the precise description of that domain. The user must realize that the final possibility distribution created by the toolkit is based on a domain of values accumulated as possibility distributions were combined in the inferencing scheme. To maintain the lingval facts temporarily, the toolkit chooses one domain and describes the resulting possibility distribution as follows in Figure 3.9.

```
lingval(tall, null, 5.2, 0).
lingval(tall, null, 5.5, 0.1).
lingval(tall, null, 5.8, 0.3).
lingval(tall, null, 5.11, 0.6).
lingval(tall, null, 6.2, 0.7).
lingval(tall, null, 6.5, 0.9).
```

Figure 3.9 Possibility Distribution of Tall and Heavy People

The 'and' operator is also used in combining fuzzy modifiers with the possibility distributions of linguistic values as described in section 3.2.2. Given the definition of "very" based on truth values and the definition of "tall," fuzzy logic is used as the basis for combining "very" AND "tall" to produce "very tall." The toolkit automatically combines data content fuzzy modifiers with linguistic values, as defined by the expert system developers, using the same fuzzy "and" operation.

To complete this discussion, the concept of a default possibility distribution needs to be introduced. Not every clause in an expert system contains data content uncertainty. Certain knowledge, such as the statement "Steve is 5'11"" is

quite precise. The possibility distribution of such a statement is shown in Figure 3.10.

```
lingval(tall, null, 5.2, 0).
lingval(tall, null, 5.5, 0).
lingval(tall, null, 5.8, 0).
lingval(tall, null, 5.11, 1).
lingval(tall, null, 6.2, 0).
lingval(tall, null, 6.5, 0).
```

Figure 3.10 Possibility Distribution of 5'11" People

It contains no uncertainty as all points are strictly within or beyond the set. This possibility distribution must, none the less, be predefined by the expert system developer.

Other expert system clauses, such as input/output functions or other builtin PROLOG functions do not contain data content uncertainty and would be difficult for the end user to predefine. Such functions are useful in executing the expert system but do not affect the certainty or uncertainty of the rules in the system. Since the toolkit requires that every clause have a possibility distribution to facilitate the inferencing logic, a generic default possibility distribution is needed to be assigned to clauses with no possibility distribution of their own. These builtin functions have been identified by the toolkit along with the default possibility distribution. The latter is described in Figure 3.11.

```
lingval(tall, null, a, 1).
lingval(tall, null, b, 1).
lingval(tall, null, c, 1).
lingval(tall, null, d, 1).
lingval(tall, null, e, 1).
lingval(tall, null, f, 1).
```

Figure 3.11 Default Possibility Distribution

Actual domain values could be substituted for the letters (a through f) if necessary. Constant degrees of membership values all equal to 1 indicate that the entire domain is within the set. Given that the toolkit uses the minimum operator of fuzzy logic in combining rule clauses, any possibility distribution combined with the default possibility distribution will result in the original possibility distribution. This assumes that the degrees of membership in all possibility distributions are in the interval [0,1] which is guaranteed by the toolkit. In this manner, the default has the effect of an identity function and does not influence the certainty of the possibility distributions, which is the desired end.

3.2.6 Further Discussion of the 'Or' and 'Not' Operators

In certain instances, the 'or' operator is useful in expert system rules. For example: "a person is big if the person is tall or the person is heavy". This can be expressed in PROLOG as:

```
big(X) :-
  tall(X).
big(X) :-
  heavy(X).
```

If Steve is tall, then the query:

```
big(steve).
```

would be satisfied by the first rule. If all that is known is that Steve is heavy, then the first rule would fail but the second rule would be satisfied and a positive result would occur. Obviously, these rules and facts could be embellished with data content and data context uncertainty modifiers, but unmodified terms will suffice for this example.

Assume that it is known that Steve is both tall and heavy. Then both rules would satisfy the question "Is Steve big?". Basic PROLOG would execute the first rule and ignore the second in this situation. The toolkit, however, uses a PROLOG builtin function called 'setof' to force the system to consider all alternatives. The possibility distribution of both rules in this example is determined and then combined using the maximum operator described by fuzzy logic. Given Figure 3.1 and Figure 3.7, which describes tall and heavy (the results of the two "big" rules), the maximum operator results are shown in Figure 3.12.

```
lingval(tall, null, 5.2, 0).
lingval(tall, null, 5.5, 0.2).
lingval(tall, null, 5.8, 0.4).
lingval(tall, null, 5.11, 0.6).
lingval(tall, null, 6.2, 0.8).
lingval(tall, null, 6.5, 1.0).
```

Figure 3.12 Possibility Distribution of Tall or Heavy People

The toolkit effectively creates a result that is the union of the two alternatives and provides a more certain fuzzy subset.

The 'not' operator proved to be an interesting case in

the toolkit inferencing logic. The use of a 'not' operator in an expert system clause requires the targetted fact to fail in order for the entire clause (including the 'not') to succeed. If a person is not short then the person is big. In PROLOG:

```
big(X) :-
  not(short(X)).
```

In order for the toolkit to properly recognize the 'not' operator, the syntax was changed to 'fnot' to avoid overloading the builtin PROLOG 'not' operator:

```
big(X) :-
fnot(short(X)).
```

The only way that big(steve) will be true is if there is no fact in the knowledge base asserting that Steve is small. In general, in order for a 'not' clause to fail (and thus be true), some point in the inferencing scheme chain must have been unsatisfied. The toolkit only assigns possibility distributions to facts or clauses that are resolved. Therefore, the toolkit must assign the default possibility distribution (previously described) to 'fnot' clauses since it is impossible to address the certainty/uncertainty of knowledge that does not exist.

The fuzzy logic 'not' operator did prove to be useful in the definition of some fuzzy modifiers. Just as the toolkit recognizes positive fuzzy modifiers such as "very", it also handles negative fuzzy modifiers like "not very". All fuzzy modifiers (positive and negative) expressed in an expert

system are dynamically defined by the toolkit. Each modifier is predescribed in the toolkit in terms of a category of uncertainty and its connotation (positive or negative). When the modifier is first used in an expert system, the toolkit automatically builds the possibility distribution of the modifier. For positive modifiers this just implies creating the possibility distribution directly from the degrees of membership of the category of uncertainty. For negative operators, the fuzzy logic 'not' operator is applied and the inverse of the degrees of membership in the uncertainty category is calculated and used in the possibility distribution. Therefore, "not very" would be created as in Figure 3.13.

```
lingval(true, not_very, 1, 1.0).
lingval(true, not_very, 2, 1.0).
lingval(true, not_very, 3, 1.0).
lingval(true, not_very, 4, 0.9).
lingval(true, not_very, 5, 0.7).
lingval(true, not_very, 6, 0.5).
```

Figure 3.13 Possibility Distribution of Not Very True

The fuzzy subset represented by this membership function is similiar to the "very" fuzzy subset except that the center or nucleus of the subset has shifted to the lower end of the domain. It should be pointed out that not all negative fuzzy modifiers begin with the word "not". Modifiers such as "sortof", "sometimes" or "a_little" have more of a negative connotation than a positive one and are treated by the toolkit in the same way as modifiers beginning with "not". Appendix B contains a list of valid negative fuzzy

modifiers along with the positive fuzzy modifiers.

To complete this section, a recommendation is made in utilizing the 'not' operator ('fnot', as recognized by the toolkit) in rule clauses. In most instances, this type of operator used with a linguistic value can be replaced with a positive linguistic value, rather than negating a negative lingustic value. In the example, it is better to express "not short" in a positive sense:

```
big(X) :-
  tall(X).
```

It is also possible, although not recommended, to include the 'not' in the definition of the linguistic value, as in "not_short". In either case, the toolkit will execute properly as long as the domain values and degrees of membership have been defined correctly and reflect the fuzzy subset.

3.2.7 Approximate Reasoning in the Toolkit

Approximate reasoning involves the translation of input natural language statements into fuzzy subsets or relations. These subsets, in the form of possibility distributions, are then combined using fuzzy logic to create others of the same and retranslated to natural language statements for output to the user. The toolkit currently does not translate natural language statements into possibility distributions. It does

however, expand the language of PROLOG to express both data content and data context uncertainty in expert systems given two new PROLOG syntax forms. It also allows for input queries containing data content uncertainty such as:

```
tall(steve):very.
```

"Is Steve very tall?". This capability will be explained further in later sections. It should be noted too, that the toolkit does not alter any input processing performed by the expert system itself. Therefore, the user may already be entering information into the expert system in natural language if the expert system has the capability of interpreting it. Certainly, a simple natural language transition network parser could be added to the toolkit as an extension for basic queries.

The second issue in approximate reasoning is the combination and creation of possibility distributions. As thoroughly described in this section, the toolkit enhances the inferencing scheme of PROLOG by providing a possibility distribution whenever input queries or rule clauses are satisfied. If the clause is resolved by a fact, a single possibility distribution is produced. For a rule to be satisfied, the toolkit must combine the possibility distributions of each term in the rule to create a single resultant possibility distribution. This is done by applying fuzzy logic operations to the possibility distributions to create one of the same. In either case, a single possibility distribution is supplied by the toolkit for output to the

user.

This raises the final, most difficult issue, that of retranslating the resulting possibility distribution into a natural language statement for output. The results of this retranslation process should be approximate consequents of the original statements, due to the use of variables containing linguistic values rather than precise numerical values [Zadeh 1974]. However, as discovered in section 3.2.3 (describing the threshold rating scheme), expressing a fuzzy subset by examining a possibility distribution is no trivial task. Making an attempt to retranslate the possibility distribution into a single fuzzy modifier would only disguise the details of the fuzzy subset. Attempting a more sophisticated translation would be beyond the scope of this project. Instead, the possibility distribution is displayed for the user's interpretation. The following sections, which describe the toolkit environment, discuss methods for understanding the output possibility distributions.

3.3 A Description of the Toolkit Environment

This section on the implementation of a fuzzy set theory approach to uncertainty describes the mechanics of executing a PROLOG expert system using the toolkit. The specific environment under which the toolkit runs is outlined. An interesting example of a very small knowledge base is introduced and will be carried through the section as the toolkit is described. The four main components of the toolkit are given: predefining linguistic values, reviewing linguistic values, setting up the expert system to run and executing the expert system within the toolkit. Finally, careful attention is given to the control and understanding of the diagnostic output, as well as interpreting the final results provided by the toolkit.

The following sections provide a thorough description of the toolkit. In addition, Appendix D contains all of the possible messages generated by the toolkit and Appendix E gives a complete listing of the output generated by the toolkit in the example to be presented. A selection of some of the actual toolkit code is included in Appendix F.

3.3.1 Toolkit Environment Specifics and an Example

The toolkit consists of a series of software programs written in C-PROLOG on the Rochester Institute of Technology Pyramid VAX machine running under the UNIX operating system. Specifically, there are eleven files containing PROLOG rules and facts:

lingset - predefines linguistic values and review.
esset - automatic setup of an expert system
expert - main inferencing scheme logic driving the execution of the expert system
setup - initializes databases and sets threshold
thresh - contains the thresholding logic
context - contains data context uncertainty logic
truth - predefined truth tables
tools - miscellaneous PROLOG tools and utilities
diagnos - contains and controls all diagnostic messages
wrapup - contains 'or' logic and final toolkit output
loadkit - loads all ten previous files

These files must be loaded into PROLOG in order to execute the toolkit. This is accomplished by loading just the loadkit file which then loads the remaining files. A listing of four of the files (expert, thresh, context and truth) can be found in Appendix F.

Before utilizing the toolkit, it is assumed that the

expert system developer has an expert system written in C-PROLOG within the same environment. The toolkit requires the expert system to be stored in one single file. This expert system may be modified by the uncertainty operators (colon and double dash) previously described. As such, the expert system would not be executable under normal PROLOG.

The toolkit supplies four main functions:

1) Predefining linguistic values in terms of a membership function of domain values and degrees of membership.
2) Reviewing linguistic values established by the expert system developer as well as fuzzy modifiers predefined in the toolkit.
3) Reading the modified expert system file and storing the expert system in a format accessible to the toolkit.
4) Executing the expert system under the enhanced inferencing scheme provided by the toolkit.

These functions are carefully examined in the coming sections. To assist in the discussion, an example of a slightly more complex situation is presented. The concepts of data content uncertainty (including both positive and negative modifiers), data context uncertainty, thresholding and combining clauses using the both fuzzy logic 'and' and 'or' operators are reviewed.

```
should_attend_college(X) :-
  interested_in(X,Y):definitely,
  rich(X)--absolutely.
interested_in(steve,dentistry):sortof.
```

```
interested_in(steve,computer_science):very_much.
interested_in(steve,mechanics):definitely.
rich(steve):very--really.
```

The rule reads: "It is absolutely true that a person X should attend college if the person is definitely interested in a subject Y and if the person is rich". The knowledge base also contains four facts: "Steve is sortof interested in dentistry", "Steve is interested in computer science very much", "Steve is definitely interested in mechanics" and "It is really true that Steve is very rich". The input query to be used is:

```
should_attend_college(steve).
```

"Should Steve attend college?".

3.3.2 Predefining Linguistic Values

In order for the toolkit to handle uncertainty in expert systems using fuzzy set theory, it is necessary to describe the fuzzy subsets used to represent the uncertainty. This is accomplished by defining membership functions of all linguistic values in the expert system. By specifying domain values and corresponding degrees of membership, the toolkit can create possibility distributions to be utilized in the inferencing engine.

The toolkit provides a simple way to predefine linguistic values once they have been identified by the

expert system developer. Any fact or clause to which a data content fuzzy modifier has been applied must be predefined to the system. Likewise, any fact or rule quantified by a data context modifier must be predefined. This assumes that these modifiers are being used to quantify linguistic values and not such things as PROLOG builtin functions.

In the example, three linguistic variables exist: "college_applicants" (linguistic value, should_attend_college), "interest" (linguistic value, interested_in) and "wealth" (linguistic value, rich). To initiate the toolkit linguistic values predefinition session, type the following in PROLOG:

```
ling_val_setup.
```

The toolkit will first prompt the user for the name of the linguistic value. It then prompts the user six times for a pair of domain values and degrees of membership. Note that all output by the toolkit is prefixed by the term "Toolkit-xxxxn", where xxxxn uniquely identifies the message (see Appendix D for a complete description of the output messages). For example:

```
ling_val_setup.
Toolkit-Input1: Enter first linguistic value
                (enter done if none):
|: rich.
Toolkit-Input3: Enter domain value number 1
|: 10000.
Toolkit-Input4: Enter degree of membership number 1
```

```
|: 0.
'
'
'
Toolkit-Input3: Enter domain value number 6
|: 100000.
Toolkit-Input4: Enter degree of membership number 6
|: 0.8.
Toolkit-Input2: Enter next linguistic value
                (enter done if none):
|: done.
```

Note that six pairs must be entered and should be specified in ascending domain value sequence. The degrees of membership must be in the interval [0,1]. The user may continue by entering another linguistic value or may complete the process by keying in the word "done" as above. This process must be done once each time PROLOG is invoked. However, it need not be repeated before every iteration of the expert system.

Assume that the membership functions of "should_attend_college", "interested_in" and "rich" are specified using the ling_val_setup function. The next section describes a function to review the possibility distributions created by the toolkit in this process.

3.3.3 Reviewing Linguistic Values and Fuzzy Modifiers

At any time after the toolkit is loaded into PROLOG, it

is possible to review any possibility distribution maintained in the system. This includes possibility distributions created during the user's setup of linguistic values. It also includes predefined linguistic values such as true and the fuzzy modifiers based on the truth concepts. The toolkit displays the possibility distribution of the linguistic values (possibly modified by fuzzy modifiers) in terms of the membership function that describes the fuzzy subset. To examine a possibility distribution in the toolkit, key in:

review.

The toolkit prompts the user for the linguistic value to be reviewed. It also asks if a fuzzy modifier should be applied to quantify the data content uncertainty. Given this input, the toolkit displays the membership function requested. For instance, the following scenerio demonstrates how the review function works in the toolkit to review the definition of "rich" and "very rich". Key in:

review.

Toolkit-Input5: Enter linguistic value to review:

|: rich.

Toolkit-Input6: Enter modifier to review:

|: null.

Toolkit-Info2:

Domain Value	Degree of Membership
10,000	0
25,000	0.1
40,000	0.3
65,000	0.5
80,000	0.6
100,000	0.8

```
Toolkit-Input7: Enter next linguistic value
                (enter s for same or d for done):
|: s.
Toolkit-Input6: Enter modifier to review:
|:very.
Toolkit-Info2:  Domain Value   Degree of Membership

                10,000         0
                25,000         0
                40,000         0
                65,000         0.1
                80,000         0.3
               100,000         0.5
Toolkit-Input7: Enter next linguistic value
                (enter s for same or d for done):
|: d.
```

It is also possible to review a linguistic value with a fuzzy modifier that is not used explicitly in the knowledge base. For instance, reviewing "quite rich" will present the appropiate membership function (the review program creates the membership functions dynamically). To display the definition of a fuzzy modifier itself, review the linguistic value "true" with the fuzzy modifier desired.

Once the toolkit outputs the membership function it prompts the user for another linguistic value to display. The user may request another linguistic value or input "s" to maintain the same linguistic value. The latter may be done to review the same linguistic value with a different fuzzy modifier, since the toolkit will prompt for the fuzzy modifier again as well. To complete the review process, a "d"

may be entered for "done."

For completeness, the should_attend_college and interested_in are reviewed.

review.

```
Toolkit-Input5: Enter linguistic value to review:
|: should_attend_college.
Toolkit-Input6: Enter modifier to review:
|: null.
Toolkit-Info2:  Domain Value   Degree of Membership

                     1                 0
                     2                 0.1
                     3                 0.3
                     4                 0.5
                     5                 0.7
                     6                 0.9

Toolkit-Input7: Enter next linguistic value
                (enter s for same or d for done):
|: interested_in.
Toolkit-Input6: Enter modifier to review:
|: null.
Toolkit-Info2:  Domain Value   Degree of Membership

                     1                 0.1
                     2                 0.3
                     3                 0.5
                     4                 0.7
                     5                 0.9
                     6                 1.0

Toolkit-Input7: Enter next linguistic value
                (enter s for same or d for done):
|: d.
```

3.3.4 Automatic Setup of the Modified Expert System

Once the linguistic values are defined to the toolkit, it is necessary to input the entire modified expert system into the toolkit. This is accomplished with the expert system file setup function. Key in:

```
exp_sys_setup(FILE).
```

where FILE is the name of the UNIX file containing the expert system. Note that this assumes that the expert system is stored completely in a single file and that the file is not a load file (a future enhancement). The toolkit reads all the rules and facts in the file. All data content and data context uncertainty fuzzy modifiers are stripped from the clauses and stored along with the facts and rules in a format recognizable to the toolkit inferencing scheme. Specifically, the exp_sys_setup function creates a database of facts containing the expert system. The facts are in the form:

```
exp_sys(Head,Body,FactMod,TruMod).
```

Where Head and Body comprise an unmodifed clause and FactMod and TruMod represent the data content and data context uncertainty modifiers, respectively. For instance, the clause

```
big(X) :-
  tall(X):very--really.
```

is stored as the fact:

```
exp_sys(big(X),tall(X):very,null,really).
```

Also, the expert system fact

rich(steve):quite--absolutely.

is represented by:

exp_sys(rich(steve),null,quite,absolutely).

Notice that data content uncertainty modifiers defined on rule goals, such as tall(X):really, are not stored as FactMod variables. Instead, the toolkit inferencing scheme interprets these modifiers within the body as it executes the thresholding logic. FactMod is used exculsively for data content uncertainty modifiers applied to expert system facts.

In addition to translating the modified expert system into exp_sys facts for the toolkit, the exp_sys_setup function also audits all facts that are linguistic values modified by a fuzzy modifier. Additional auditing of linguistic values in rule goals is done in the inferencing scheme. If a linguistic value is found anywhere in the expert system that was not predefined in the ling_val_setup routine, an error message is displayed to the user (see Appendix D). The user should define the missing linguistic value using the ling_val_setup function and rerun the exp_sys_setup process.

Finally, the toolkit requires some additional control information, which is input during exp_sys_setup. The user is presented two questions:

Toolkit-Input8: Display diagnostics as toolkit runs (y/n)?

The user must respond with a 'y ' (yes) or a 'n' (no). A

positive response will cause the toolkit to display various messages to the user as it executes the expert system. This allows the user to follow the uncertainty inferencing logic in the toolkit including the combination of possibility distributions using fuzzy logic operations and the threshold values determined by the toolkit. The diagnostic output is described in more detail in section 3.3.6. This type of output can be eliminated by answering no to this question.

The second question is:

Toolkit-Input9: Invoke thresholding in toolkit runs (y/n)?

The user must respond with a 'y ' (yes) or a 'n' (no). This allows the user to control whether or not a threshold is used. If a threshold is desired (a yes answer), then each time the expert system is executed, the user will be required to specify the threshold value. If thresholding is not desired (a no answer), the exp_sys_setup function will assign a default threshold of zero. The user will not be prompted for a threshold value on each iteration of the expert system. Note that a zero threshold is an absolute low threshold. This will allow the inferencing method to satisfy even the weakest clauses. It provides the expert system with the loosest possible interpretation of fuzzy modifiers, which may not be desirable. It is suggested that the thresholding mechanism be utilized when executing expert systems using the uncertainty toolkit.

For the sake of the example, assume that the should_attend_college knowledge base is stored in a file

called college. Then,

```
exp_sys_setup(college).
```

will establish the facts and rules as the exp_sys facts that follow.

```
exp_sys(should_attend_college(X),
        [interested_in(X,Y):definitely,rich(X)],
        null,absolutely).
exp_sys(interested_in(steve,dentistry),null,sortof,null).
exp_sys(interested_in(steve,computer_science),null,
        very_much,null).
exp_sys(interested_in(steve,mechanics),null,definitely,
        null).
exp_sys(rich(steve),null,very,really).
```

Also assume that both diagnostics and thresholding will be included in the execution of the knowledge base.

3.3.5 Running the Expert System Using the Toolkit

Given that the ling_val_setup and exp_sys_setup functions have been completed successfully, it is appropiate to begin the execution of the expert system within the toolkit. This is done simply by keying:

```
run(EXPSYS).
```

where EXPSYS is the clause that invokes the expert system. To run the example, type:

```
run(should_attend_college(steve)).
```

If thresholding was requested in the exp_sys_setup, the toolkit will prompt the user for a threshold. This is a value between 0 and 50, with 50 being the most stringent threshold. The higher the threshold, the tighter the uncertainty interpretation becomes. A value of 50 requires an exact match on fuzzy modifiers for a clause to be satisfied. For instance, the knowledge that "Steve is really tall" would not satisfy the query "Is Steve very tall?". The knowledge base must contain exactly the knowledge "Steve is very tall" for a match to occur. On the other hand, a low threshold such as 0, allows for a very loose interpretation in matching fuzzy modifiers. In this case any knowledge that "Steve is tall," even the knowledge that "Steve is sortof tall," would match the query "Is Steve very tall?".

Once the threshold is set, the toolkit executes the expert system. Any input required by the expert system and all output generated by the expert system will by handled normally. However, if diagnostics were requested, additional information from the toolkit will be provided. Note that this may involve a substantial amount of output that may be confusing to the user. Therefore, diagnostics should be included with caution. All diagnostic output is labeled with a "Toolkit-DiagN" prefix (see Appendix D).

Whether or not diagnostics are displayed, the toolkit always provides a final result. This represents a possibility distribution describing the certainty/uncertainty of the expert system result. Section 3.3.7 will examine the toolkit

result in further detail. Since the toolkit has modified the inferencing method of PROLOG, it is quite possible that the results of the expert system run under the toolkit may vary from an unmodified expert system. One might compare the results from both the modified expert system run under the toolkit and the unmodified expert system run under PROLOG (without the toolkit). Given the ability to express data uncertainty and reason under uncertainty, the toolkit should provide more reasonable results.

The toolkit allows one other option in executing the expert system. It is possible to specify a fuzzy modifier on the input clause. The toolkit will use this fuzzy modifier as an additional requirement in satisfying an expert system. The thresholding logic will be applied to determine if the resultant possibility distribution of the query is sufficient to satisfy the possibility distribution of the modified input query. An example would be:

```
run(should_attend_college(steve):really).
```

"Should Steve really attend college?". Once the toolkit determines the possibility distribution of should_attend_college(steve), it compares it to the possibility distribution of the entire query should_attend_college(steve):really. If it is within the acceptable threshold, the result is positive; otherwise the query fails.

This type of query utilizes the same syntax for data content fuzzy modifiers presented earlier (the colon).

However, since the thresholding logic applies only to data content uncertainty at this time, the toolkit does not recognize the second new syntax presented (the double dash) for data context uncertainty. Therefore, a syntax error will occur if the following query is attempted:

```
run(should_attend_college(steve)--really).
```

"Do you really believe that Steve should attend college". This type of query could be added to the toolkit as an enhancement at a later time.

3.3.6 Understanding the Diagnostic Output

This section describes the diagnostic output generated by the toolkit as it executes the should_attend_college example. It is a rigorous explanation of this type of output and could be skipped if there is no interest in reviewing the diagnostics. The next section includes an interpretation of the final toolkit result, which is more important.

The diagnostic output produced by the toolkit leads the user through the inferencing logic as the toolkit progresses through the expert system. The example will be helpful in describing this process and the corresponding output. Please refer to Appendix E throughout this section, as it contains a complete listing of the actual output.

Recall the should_attend_college rule presented in section 3.3.1. The question, "Should Steve attend college?"

is entered as:

run(should_attend_college(steve)).

The toolkit begins by requesting a threshold value, since this was specified in the exp_sys_setup process:

Toolkit-Input10: Enter threshold value (0,50),
where 50 is the most stringent value:

Assume a threshold of 30 is entered, which will provide an average interpretation of the uncertainty in the knowledge base. The should_attend_college clause is matched with the first rule in the knowledge base and the variable X is instantiated to steve. In order to satisfy this rule, the goals "interested_in" and "rich" must be resolved.

To satisfy the "interested_in" clause, the fact

interested_in(steve,dentistry):sortof

is first examined. The toolkit defines this fact with the data content fuzzy modifier "sortof" and recognizes a null or nonexistant data context fuzzy modifier (see output). It used the possibility distribution of "interested_in" and the definition of "sortof" to create the resulting degrees of membership shown (note that this definition was done in the exp_sys_setup process). It must then decide if the fact that Steve is sortof interested in dentistry satisfies the clause that Steve must be definitely interested in something (Y). It dynamically defines the clause interested_in with the data content modifier "definitely" to produce temporary degrees of membership (see output) used in the thresholding logic.

To compare "sortof interested in dentistry" to

"definitely interested in Y", the threshold rating scheme is applied, which produces ratings of 2.4 and 12.6 respectively. The difference is 10.2 which maps to a threshold value of 23 (Appendix C). The threshold entered was 30; therefore, the fact does not satisfy the clause. The user will not allow such a loose interpretation of the modifier "definitely" such that it would accept a fact specified with a negative modifier like "sortof". The toolkit responds by indicating that the threshold must be less than or equal to 23 in order for such a fact to be used and indicates that the threshold check fails on the fact definitely interested_in (see output).

Since the first interested_in fact could not satisfy the rule clause, the next fact (Steve is interested in computer science very much) is investigated. The toolkit recalls and displays the definition of "very_much interested_in". It also notices that no data context modifier has been applied to the fact (see output). It again defines the degrees of membership of "definitely interested_in" temporarily (see output) for the threshold check. The rating for "definitely interested_in" has not changed from 12.6. The rating for "very_much interested_in" is calculated at 9.9 with a difference of 2.7. The toolkit indicates that a acceptable threshold (from the mapping function) must be less than or equal to 41 (see output). Since the user's threshold is 30, the toolkit resolves the goal with

interested_in(steve, computer_science):very_much

and can now continue with the next goal.

The toolkit processes the rich(steve) goal in the same fashion. The fact rich(steve):very--really is defined. While "very rich" was predefined, the toolkit must now also apply the data context modifier "really". The result is a set of degrees of membership representing a fuzzy subset with less uncertainty (see output). This possibility distribution calculates a rating of 13.1. The toolkit then defines the linguistic value "rich" with no data content modifier, since none was specified on the rule goal (see output). A rating of 5.3 is calculated for "rich". Comparing the two ratings, the result is negative. Not only is it true that Steve is rich, it is really believed that Steve is very rich. The fact more than satifies the clause and the toolkit makes this observation (see output).

Now that both goals in the rule have been satisfied, the toolkit combines the possibility distributions of the facts that satisfied each goal to produce a possibility distribution for the rule. This is done using the fuzzy logic "and" operator, which determines the minimum value for each corresponding degree of membership. The toolkit describes this explicity (see output) by displaying both possibility distributions and the resultant degrees of membership. Note that the possibility distribution used for each goal is that of the fact satisfying the goal and not that of the modified clause. For instance, the toolkit utilizes the possibility distribution for "very_much interested_in computer science"

rather than the temporary possibility distribution created to define "definitely interested_in Y". The latter was used solely for thresholding as an additional requirement in satisfying the goal.

Once the possibility distribution of the rule has been created, the toolkit checks for a data context modifier specified on the rule. In this case, the data context modifier "absolutely" has been used. This is applied to the rule possibility distribution, again reducing the uncertainty of the fuzzy subset reflected in the degrees of membership (see output). Finally, the toolkit examines the input clause, defining should_attend_college with a null data content modifer (since none was specified on input). The thresholding logic is used once more to compare the resultant possibility distribution (15.8 rating) with the input clause (5.5 rating). The resultant clause more than satisfies the input query, noted by the toolkit (see output).

This might be the end of the expert system execution. However, the toolkit forces it to process every possible path to implement the fuzzy "or" operation. In this case, the knowledge base contains a third "interested_in" fact that could satisfy the rule. Steve should absolutely attend college if he is interested in computer science very much or he is definitely interested in mechanics (and he is rich). The toolkit examines this third "interested_in" fact, defining it with the data content modifier "definitely" (see output). Also defining the clause "definitely interested_in

Y", the toolkit observes that an exact match is made (see output) since both the fact and the clause are modified by "definitely". The "rich" goal is satisfied in the same manner as above, and fuzzy logic is used to combine the two goals. The result is slightly different from the previous iteration due to the difference in the "very_much" modifier (Steve is very much interested in computer science) and the "definitely" modifier (Steve is definitely interested in mechanics). The final result is also slightly different once the "absolutely" data context modifier is applied (see output). This new possibility distribution also more than satifies the input query (see output).

What remains then, are two possibility distributions that more than satisfy the input query. Since this is an "or" situation (stated above), the fuzzy logic "or" operator is applied. The toolkit responds by displaying both intermediate results and the final degrees of membership after taking the maximum of the corresponding degrees of membership (see output one last time).

PROLOG responds with a 'yes' to indicate that the knowledge base successfully satisfied the input query. The next section is helpful in interpreting the final degrees of membership produced by the toolkit.

3.3.7 Interpreting the Results

In addition to the output normally produced by an expert system, the uncertainty toolkit produces a list of six final degrees of membership. The result for the should_attend_college example was:

Toolkit-Info5: Final degrees of membership:

0 0 0 0 0 0.8

Section 3.3.6 discussed in detail how these results were created, but it is still necessary to describe what the result means.

To begin, an observation concerning the final output is made. The result is simply a list of six degrees of membership. The toolkit does not present the degrees of membership in the form of a membership function; the result is lacking in domain values. As mentioned previously, the toolkit does not attempt to maintain the domain values of the fuzzy subsets as they are combined. Therefore, the user must realize that the domain upon which the result is based, is a specific combination of the domains in the expert system. In this case, the domain is made up of "interests" and "wealth" domain values.

Given that the final output is simply a list of six degrees of membership based on a combined domain of values, there are a number of characteristics to examine to determine the essence of this final fuzzy subset. A good way to interpret the degrees of membership of a fuzzy subset is to

consider its characteristics in terms of data content and data context uncertainty. Characteristics such as the peak or center of the fuzzy subset, the width of the boundary area and the magnitude of the degrees of membership within the boundary area are considered in this fashion.

Data content uncertainty describes the extent to which something is, or has or should occur. Not only is the query asking if Steve should attend college, but to what extent. The first clue to this answer is the peak of a fuzzy subset, or the location of the center of the fuzzy subset relative to the domain values. Assuming ascending domain values, if the peak of the fuzzy subset is to the right or high end of the domain, then the result is positive. The degrees of membership are also in ascending order, corresponding to the domain values. Likewise, a low end peak or descending degree of membership values, produces a negative result. Fuzzy subsets that peak neither at the high end nor the low end may have a preference one way or the other, but the final result is not as clear cut as the two described (positive or negative). In the example, the center of the fuzzy subset is to the right. The degrees of membership are ascending (although only slightly) to the high domain values. Therefore, one can conclude that the answer to the question "Should Steve attend college" is yes. But to what extent?

Data content modifiers have the effect of shifting degrees of membership to the right for positive modifiers and to the left for negative modifiers. The stronger the

modifier, the greater the shift is. This is due, for example, to the fact that a positive modifier describes a fuzzy subset that has shrunk, causing lower domain values to fall entirely outside the fuzzy subset and increasing the distance from the inner boundary edge to the remaining boundary area points. Therefore, it is possible to examine the degrees of membership of a fuzzy subset and determine the extent to which it has been modified. In the example, the degrees of membership have been shifted radically to the right, indicating a strong modifier. In fact, all but one domain point lies within the fuzzy subset at all (in the boundary area). One might correlate these degrees of membership to the degrees of membership of the fuzzy modifiers predefined by the toolkit, to produce a fuzzy modifier that expresses the extent of the fuzzy subset [Negoita 1985]. The fuzzy modifier "positively" would best describe the extent to which Steve should attend college (Steve should positively attend college). It most closely matches the resultant degrees of membership (see Appendix A and Appendix B). The toolkit could have provided a corresponding fuzzy modifier in place of the degrees of membership as output. However, it was felt that such a rough translation would mask the other characteristics of the fuzzy subset.

Data context uncertainty concentrates on the degree of belief or confidence in a fuzzy subset; how much does one believe that the result is true. It is known that Steve should positively attend college. What is not known is the

level of confidence the knowledge base has in this statement. Data context modifiers affect the boundary area of a fuzzy subset. A high degree of confidence (or a stong, positive data context modifier) decreases the boundary area of the fuzzy subset. The subset becomes more certain. On the other hand, a negative data context modifier will increase the boundary area, creating more uncertainty. Examining the boundary area of a fuzzy subset will indicate the degree of certainty or confidence in a fuzzy subset. The should_attend_college example has a very small boundary area. In fact, most domain points lie completely outside the outer boundary (due in part by the strong effect of the data content modifier). Even though none of the domain points are strictly within the fuzzy subset, it is obvious that the fuzzy subset is quite small and concise with a low degree of uncertainty. It is therefore possible to state that there is great certainty that Steve should positively attend college.

One final characteristic of the fuzzy subset to observe, is the magnitude of the degrees of membership within the boundary area. Recall that the greater the degree of membership, the closer the domain point was to the inner edge of the boundary (closer to complete membership). Both data content and data context uncertainty affect this magnitude. Data content uncertainty reduces degrees of membership as it shifts the inner boundary edge away from the domain points. Data context uncertainty shifts the inner boundary edge to the left as it decreases the total boundary area, thus

increasing the degrees of membership. In either situation, the degrees of membership of domain points within the boundary area should be considered. Given two fuzzy subsets, both containing the same boundary placement and width (same number of domain points within the boundary area), the fuzzy subset containing degrees of membership closer to one or the other boundary edge might have more certainty than the second subset containing degrees of membership spread evenly across the boundary. A slight change in the confidence of the former fuzzy subset would greatly reduce its uncertainty. A scheme to measure the uncertainty of such fuzzy subsets would be worth pursuing as an enhancement to the uncertainty toolkit.

Chapter 3 has presented the fuzzy set theory uncertainty toolkit implemented for this thesis. The representation of uncertainty and the uncertainty reasoning mechanism utilized by the toolkit have been covered. The specific environment under which the toolkit executed was described thoroughly with the help of small knowledge base. From this example, it was demonstrated that the toolkit generated a response to an input query, which, after some interpretation, provided a greater sense of both the data content and data context uncertainty specified in the knowledge base. Such a result could not have been provided without the use of fuzzy modifiers and the uncertainty toolkit.

Chapter 4

Results of the Toolkit and Conclusions

Different approaches to dealing with uncertainty in expert systems and the specific implementation of a fuzzy set theory method have been presented. This chapter draws some conclusions about the fuzzy set theory implementation as well as the comparison of it to other approaches. The test cases used to verify the fuzzy set theory uncertainty toolkit are reviewed. Some of the problems encountered during the implementation are also discussed. An analysis of the final results, given the limitations described, is presented along with the final conclusions of the thesis.

4.1 Test Cases

The first three functions, ling_val_setup, exp_sys_setup and review, proved to be rather straight forward to test. All audit conditions were exercised in predefining linguistic values, reviewing linguistic values and establishing the expert system to be run under the toolkit. A multitude of linguistic values were defined and reviewed using many different fuzzy modifiers. In the expert system, all possible configurations of rules and facts modified by both data content and data context fuzzy modifiers were tested. The

resulting ling_val and exp_sys database facts were carefully examined.

The fourth function of the toolkit, the "run" command, which executes an expert system, required a more detailed test plan. Essentially two types of test cases were developed to test the main toolkit run logic. A large base of simple single fact or single rule cases were used to test the basic concepts. These were critical in testing the system as it was being built as well as ensuring that the logic remained intact as changes were made. The second type of test case involved a smaller set of more complex knowledge base systems that verified the logic of the toolkit as a whole. This too was useful in checking that a change made to one portion of the toolkit did not adversely effect any other aspect of the system. Both types of test cases are discussed in further detail.

A substantial set of single facts and rules were defined early in the implementation of the toolkit. Fourteen facts concerning "tall people" were identified, such as:

```
tall(john).
tall(carl):sortof.
tall(joe)--absolutely.
tall(larry):quite--really.
```

This base of facts represented every combination of data content and data context uncertainty modifiers using different instances of the modifiers for effect. The intent was to ensure that the toolkit was applying the modifiers to

the facts correctly and to the proper extent (different modifiers have varying degrees of effect on a linguistic value). From the fourteen test case facts, six cases were singled out. They included:

```
tall(john).
tall(sue):very.
tall(steve)--really.
tall(james):very--really.
tall(joe)--absolutely.
tall(jack):very--absolutely.
```

These cases represented one data content modifier and two data context modifiers. Each of these facts were tested using the same three queries:

```
tall(x).
tall(x):very.
tall(x):unbelievably.
```

where x was replaced with each name (john, sue, etc.). This type of test allowed for a realistic comparison of the results of the toolkit relative to the other five cases and gave an excellent perspective of the results of the toolkit in general.

The 'big' rule described in section 3.2 was used as a basis for a set of simple test rules. Sixteen cases were identified that were modifications of the original rule. Although not completely exhaustive, the cases provided a wide range of variations including data content modifiers, data context modifiers, the use of the fnot operator and examples

of both the 'and' and the 'or' operators. These test cases were particularly helpful in testing and maintaining the fuzzy logic operations.

In addition to this base of test facts and rules, two more complex examples were used. The most obvious case is the should_attend_college example used extensively in Section 3.3. This knowledge base became a tool to test the interactions of a number of useful cases and situations. These included data content and data context uncertainty (both positive and negative modifiers), numerous thresholding situations and the use of both the "and" and "or" fuzzy logic operators. A second medical expert system, inspired by Ken Bowen [Bowen 1985], utilized many of the same concepts in addition to the fact that this expert system itself interfaced with the user (requesting input). In addition, the expert system contained a number of other PROLOG builtin functions (including cuts), which the toolkit had to handle.

Both the base cases and the small knowledge bases described provided an effective and thorough means of verifying the concepts of fuzzy set theory as applied to the problem of uncertainty reasoning in expert systems utilizing the toolkit.

4.2 Problems and Limitations

The primary difficulties of creating the fuzzy set

theory uncertainty toolkit centered around dealing with the fuzzy subsets. The definition, manipulation and interpretation of fuzzy subsets provided ample challenges to overcome. As a method of representing uncertainty in expert systems, fuzzy subsets introduced a level of complexity and apprehension of their own.

The definition of fuzzy subsets was the first hurdle. Although the concept of fuzzy subsets was not difficult to understand, attempting to define the membership functions of certain linguistic values proved to be a challenging task. To begin with, the number of domain points chosen to define a fuzzy subset was limited to six. While this made the implementation simplier, it severely restricted the description of both domains and fuzzy subsets. There is little doubt that a larger domain set would have allowed a richer description of domains in general and fuzzy subsets specifically.

Additional problems arose in defining membership functions. One might ask the question, "What constitutes a good membership function?". Assuming a well defined domain, a wide range of domain values were typically chosen. The resulting degrees of membership were consistently spread out across the domain. Thus there was very little variability among the linguistic values. This may not be a real problem, but it made the results quite predictable. It also meant that as soon as certain data content modifiers were applied (eg. "very"), the inital degrees of membership were frequently

lost by the effects of the minimum function.

A more difficult problem arose when defining linguistic variables which do not have numerical domain values to relate to (the domain was not well defined). In these cases, the domain values were assigned a sequential number, in ascending order. These values were meaningful only in relation to the other values in terms of magnitude.

Manipulating the possibility distributions once they were defined from the membership functions did not present any serious problems (with one exception). This was due in part to the consistency in arity (the number of degrees of membership) enforced by the toolkit. Using the minimun and maximum operations to implement the 'and' and 'or' operators was both well founded and proved to create reasonable results. The only exception was the use of the minimum operator in applying data context modifers to rules. The literature, [Negoita 1985] and [Kandel, Hall, Szabo 1986], suggested that a rule of implication with the minimun function be utilized:

$$f_{\text{if R1 then R2}}(t) = \min(f_{R1}(t), f_{R2}(t))$$

The results of such an implementation were counter-intuitive. When expressing confidence in a rule such as:

```
big(X) :-
  tall(X)--absolutely.
```

the possibility distribution of the goal ("if X is tall") is combined with the possibility distribution of the head of the

rule ("then it is absolutely true that X is big"). The results of the minimum function were not reasonable. They did not reduce the uncertainty of the fuzzy subset, which is what one would expect when using the "absolutely" modifier in expressing confidence. Instead, certain heuristics had to be built to handle data context modifiers. The logic essentially reduced the boundary area of the fuzzy subset by excluding some domain value points from the fuzzy subset entirely, including others completely within the fuzzy subset and increasing the degrees of membership of those points remaining in the boundary area. This turned into a non-trivial task and required several iterations to complete.

The remaining problem in working with fuzzy subsets, was the issue of interpretation. This surfaced in two instances. In the threshold logic it was necessary to compare two possibility distributions. Therefore it was necessary to describe or rate each possibility distribution such that they could be compared. The more obvious case of interpretation was the need to describe the final output of the toolkit in some reasonable fashion. How does one describe the fuzzy subset like the one produced as the result of the should_attend_college example?

Identifying certain characteristics of a fuzzy subset was helpful in both interpretation cases. The challenge remained however, to balance the effects of data content and data context uncertainty and to provide an interpretation that was sensitive enough to the many variations of fuzzy

subsets. Developing the rating scheme used to compare possibility distributions for the purposes of thresholding, was one such challenge. Developing a translation of the resulting degrees of membership of the final toolkit output into an English like translation, was another.

The rating scheme awarded points for certain favorable conditions, which seemed effective. The mapping function to relate the differences between two possibility distributions to an acceptable threshold value was expanded from an original set of ten valid thresholds to the current fifty values to increase the sensitivity of the toolkit to fuzzy subset variations.

The decision not to attempt a translation of the final toolkit result into an English adjective (one of the set of valid modifiers) was difficult. The rating scheme used in the threshold logic could have been used to calculate a value which could have been mapped to a modifier. However, it was felt that too much of the nature of the fuzzy subset would have been lost if only a rough English interpretation of the output was displayed. Such a translation would be a simple enhancement to the toolkit, although not necessarily recommended (at best it would provide "friendliness").

Given that the toolkit does not attempt to interpret the final results, some assistance for the user is still required. Section 3.3.7 detailed a number of characteristics and observations concerning fuzzy subsets which can be made. Using these techniques, the user is able to derive a

reasonable conclusion regarding the certainty or uncertainty of the result generated by the expert system.

In terms of interpreting fuzzy subsets, increasing the arity of the domain set could be two sided. A larger domain may make the nature of the fuzzy subset more apparent. On the other hand, it may increase the confusion, due to the greater number of variations possible. In either case, a more sophisticated interpretation scheme (expanding the existing rating logic) would improve the thresholding mechanism as well as allowing for a more specific English translation of the final output. The latter could certainly enhance the users understanding of the toolkit results.

4.3 An Analysis of the Results

Analyzing the results of the fuzzy set theory uncertainty toolkit presents two perspectives: how well the concepts of fuzzy set theory were implemented ("did it work ?") and how useful the toolkit is in dealing with uncertainty in expert systems ("should the toolkit be used?").

The test cases described and executed are evidence of the fact that the toolkit processed the logic correctly. Linguistic values were successfully established as possibility distributions. Two new syntax forms allowed data content and data context uncertainty to be defined in the knowledge base and applied to possibility distributions. The

thresholding technique used to vary the level of tolerance to uncertainty allowed by the toolkit, provided a great deal of flexibility in executing an expert system. This proved to be an excellent tool in testing and experimenting with the toolkit. The fuzzy logic operators of 'and', 'or' and 'not' were implemented effectively. Finally, the rating scheme utilitized to interpret possibility distributions for the threshold logic provided a reasonable (although unsophisticated) assessment of fuzzy subsets. Along with the threshold mapping function, the rating logic was sensitive to slight differences in the membership functions of fuzzy subsets. Although this method of interpretation was useful in checking thresholds, it did not seem reasonable to use the same technique in providing a rough English translation of the output of the toolkit. Such an interpretation would have lost the effect of displaying the actual membership function of the final fuzzy subset.

The test cases used were not very large or complex. Certainly, real life expert systems are substantially more involved. However, the intent of this thesis was to demonstrate the concepts of fuzzy set theory as applied to the problems of uncertainty reasoning in expert systems. To this end, the tests served the purpose well. The actual fuzzy subsets created by the toolkit, as described by the final degrees of membership, were intuitive. Given the level of uncertainty within the knowledge base, the results accurately reflected the data content and data context uncertainty

inherent in the answer. Modifying the fuzzy modifiers in the knowledge base had the expected effects on the resultant fuzzy subset.

To determine whether the toolkit should be utilitzed with a particular expert system (large or small), a comparison must be made between the amount of overhead required by such a toolkit versus the expected benefits provided. Before any expert system can be executed within the toolkit, the linguistic values must be identified and predefined. This can be a difficult and lengthy process as previously described. Once this has been completed and the expert system is actually run using the toolkit, the user may notice a slight degradation in response time, compared to running the same expert system (void of fuzzy modifiers) outside the toolkit. This is due to the additional logic that the toolkit must perform to handle the uncertainty. Setting and using thresholds effectively may be difficult for the unaccustomed user. Likewise the final degrees of membership could be confusing to users who are unfamiliar with the concepts of fuzzy set theory.

On the other hand, there are an equal number of reasons to take advantage of the toolkit. The obvious reason is to represent the uncertainty in the knowledge of experts. Experiential knowledge is inherently fuzzy, frequently based on "rule of thumb" or "best guess" information gained only through experience. All of this fuzzy knowledge can be effectively represented and used in the toolkit inferencing

scheme. The actual process of identifying and defining linguistic values can often lead to further clarification or even new knowledge that pertains to the expert system. Using the toolkit with different thresholds can provide interesting results. A high threshold may provide a very precise result forcing the fuzzy modifiers to match exactly. A low threshold will allow much more leniency in satisfying clauses, often creating quite a different result. The toolkit can assist the user in choosing an appropiate threshold by displaying acceptable threshold values as it executes as part of the diagnostics. The toolkit can be run with or without the diagnostic listings providing even greater flexibility.

Applying fuzzy modifiers, defining linguistic values, executing an expert system within the toolkit and understanding the toolkit output are all activities which are refined through experience. The more the toolkit is utilized, the greater the benefit will be to the user. Once the learning curve is reached, the user will find that using the toolkit on an expert system will improve the information generated due to the inclusion of fuzzy infomation. The expert system will behave more as a human expert would.

One final word of caution is advisable. While the toolkit allows for the definition of uncertain knowledge, this capabililty should be used wisely. Expressing uncertainty about information should not be used as an excuse for a lack of information. The knowledge engineer working with the expert, is responsible for gathering all the

relevant detailed and exacting information that the expert has in the area of expertise. On the other hand, while the intent may be to obtain as much data as possible, over precision can result in a lack of relevancy. It is important for the knowledge engineer to recognize this delicate balance between precision and relevancy. Assuming that the expert is not masking an unwillingness to participate by being intentionally vague or imprecise, the knowledge engineer will find that a good deal of the information the expert has, is in fact, uncertain or fuzzy. Therefore, the representation of uncertainty in the knowledge base becomes an essential element in creating effective expert systems.

4.4 Conclusion

This thesis project carefully examined the five approaches to the problem of dealing with uncertainty in expert systems that are prevalent today. One method, the use of fuzzy set theory, was implemented in an uncertainty toolkit to handle uncertainty in other expert systems.

The comparison of the alternatives demonstrated that no one method dominates in all areas of uncertainty reasoning. Rather, a combination of methods may provide the maximum benefits in maintaining the credibility of an expert system.

The implementation of the fuzzy set theory uncertainty toolkit generated a number of insights. By far, the greatest difficulties in utilizing fuzzy set theory were in defining fuzzy subsets in terms of membership functions and reporting the same in a natural language. Once these problems were overcome, it became obvious that the underlying representation of uncertainty in fuzzy set theory was quite powerful. Unlike any other of the alternatives, fuzzy set theory allowed for the representation of both data content AND data context uncertainty using English terms (fuzzy modifiers) and not meaningless numbers. This capability is not parallelled in any other approach.

The toolkit, as written for the thesis, can be a useful tool for handling uncertainty in expert systems using fuzzy set theory. Several enhancements to the toolkit have been mentioned previously in this document. They focus primarily

on the interface to the user, from inputting queries with data context fuzzy modifiers to outputting meaningful remarks concerning the certainty of the expert system result.

Besides the knowledge that has been gained in completing this thesis, the greatest benefit I have received is an excitement to pursue and to follow the advancements in the resolution to the problem. This is true not only in the field of fuzzy set theory, but in context of the other approaches as well. Undoubtably, the next decade will see great advancements in many areas, which will bring the problem of dealing with uncertainty in expert systems to a closer approximation of human thinking.

Bibliography

Adams, J.B. (1984). Probabilistic Reasoning and Certainty Factors, in Rule Based Expert Systems. Buchanan, Shortliffe (eds). Addison-Wesley Publishing Company.

Bacon, J.B. (1986). Emulation of Human Treatment of Uncertainty in AI Systems. Term paper in Expert Systems course, Rochester Institute of Technology, October 1986.

Baldwin, J.F. (1985). Fuzzy Sets and Expert Systems, Information Science, Vol 36, No 1 & 2. July, August, 1985.

Baldwin, J.F. and Zhou, S.Q. (1984). A Fuzzy Relational Inference Language, Fuzzy Sets and Systems, Vol. 14, No. 2, November 1984.

Bowen, K. A. (1985). A Simple Expert System's Shell, IEEE Presentation on Expert Systems and PROLOG, 1985.

Buchanan, B.G. and Shortliffe, E.H. (1984). A Model of Inexact Reasoning in Medicine in Rule Based Expert Systems. Buchanan, Shortliffe (eds). Addison-Wesley Publishing Company, 1984.

Cheng, Y. and Kashyap, R.L. (1986). Study of Different Methods for Combining Evidence. SPIE Vol 635, Applications in AI III, 1986.

Chong, C.Y. and Fung, R.M. (1985). Metaprobability and Dempster-Shafer in Evidential Reasoning. Presented at the Uncertainty Workshop Preceeding the AAAI Conference, Philadelphia, PA, 1985.

Cohen, M. S. (1985). A Framework for Non-Monotonic Reasoning about Probability Assumptions. Presented at the Uncertainty Workshop Preceeding the AAAI Conference, Philadelphia, PA, 1985.

Cohen, P.R and Sullivan, M. (1984). An Endorsement-Based Plan Recognition Program, Department of Computer and Information Science, University of Massachussetts at Amherst, COINS Technical Report 84-33, December 1984.

Cohen, P.R. (1985). Expert Systems - Theory & Practice, National Technology University seminar, May 20 - 24, 1985.

Cohen, P.R. (1986). Uncertainty & Credibility, National Technology University lecture, January 9, 1986.

Gaines, B.R. (1977). Foundations of Fuzzy Reasoning in Fuzzy Automata & Decision Processes. Gupta, Saridis, Gaines (eds). North Holland, 1977.

Gaines, B.R. and Shaw, M.L.G. (1985). From Fuzzy Logic to Expert Systems. Information Science, Vol 36, No 1 & 2. July, August, 1985.

Gordon, J. and Shortliffe, E.H. (1984). The Dempster-Shafer Theory of Evidence in Rule Based Expert Systems. Buchanan, Shortliffe (eds). Addison-Wesley Publishing Company.

Henrion, M. and Wise, B.P. (1985). A Framework for Comparing Uncertain Inference Systems to Probability. Presented at the Uncertainty Workshop Preceeding the AAAI Conference, Philadelphia, PA, 1985.

Kandel, A., Hall, L.O. and Szabo, S. (1986). On the Derivation of Memberships for Fuzzy Sets in Expert Systems. Information Science, Vol 40, No 1, November 1986.

Lemmer, J.F. (1985). Confidence Factors, Empiricism and the Dempster-Shafer Theory of Evidence. Presented at the Uncertainty Workshop Preceeding the AAAI Conference, Philadelphia, PA, 1985.

Michie, D. (1981). Introductory Readings in Expert Systems, Gordon and Breach.

Meyer, P.L. (1970). Introductory Probability and Statistical Applications, Addison-Wesley Publishing Company, 1970.

Mostow, J. (1985). IEEE Workshop on Principles of Knowledge Based Systems; A Personal Review, ACM SIGART No 92, April 1985.

Neapolitan, R.E. (1986). A Comparison of the Mycin Model for Reasoning Under Uncertainty to a Probability Based Model. SPIE Vol 635, Applications of AI III, 1986.

Negoita, C.V. (1985). Expert Systems and Fuzzy Systems, Benjamin/Cummings Publishers.

Smith, E.E. (1985). Cognitive Psychology. Artificial Intelligence, Vol 25, No. 3, March 1985.

Zadeh, L.A. (1974). Calculus of Fuzzy Restrictions in Fuzzy Sets & Their Applications to Cognitive & Decision Processes. Zadeh, Fu, Tanaka & Shimura (eds). Academic Press, Incorporated, 1974.

Zadeh, L.A. (1978). PRUF-A Meaning Representation Language for Natural Language. International Journal of Man-Machine Studies, Vol. 10, No 4. pp 395-460, 1978.

Zadeh, L.A. (1979). A Theory of Approximate Reasoning. Machine Intelligence Vol 9. Hayes, Michie & Mikulich (eds), 1979.

Zadeh. L.A. (1984). Making Computers Think Like People, IEEE Spectrum, Vol 21, No 8, August 1984.

Appendix A: Truth Definitions

The uncertainty toolkit includes a definition of truth as a baseline for defining fuzzy modifiers. Each fuzzy modifier is assigned a category of truth that represents the degree to which it modifies the linguistic value. The categories are described below as membership functions on a domain of truth (domain values 1 through 6 along the left border). The degree of membership for each domain value is listed. The membership function of "true" (without a modifier) is category 1.

C A T E G O R I E S

	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	0.2	0.1	0	0	0	0	0	0	0	0
3	0.4	0.3	0.2	0.1	0	0	0	0	0	0
4	0.6	0.5	0.4	0.3	0.2	0.1	0	0	0	0
5	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	0	0
6	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1

Appendix B: Valid Fuzzy Modifiers

The toolkit defines a list of valid fuzzy modifiers used in expert systems to express data content and data context uncertainty. There are two types of modifiers: those that relate extent and those that relate frequency. Both types are defined by an uncertainty category described in Appendix A. One or more "not" fuzzy modifiers may also be identified. These modifiers are represented in the toolkit as the inverse of the degrees of membership in the corresponding category.

E X T E N T M O D I F I E R S

Category	Modifier	"Not" Modifier
1	true	not_true
		false
2	truely	not_truely
	mostly	almost
3	really	not_really
	fairly	kindof
4	quite	not_quite
	quite_abit	sortof
	generally	
5	alot	not_alot
	basically	a_little
6	very	not_very
	very_much	not_very_much
	very_likely	somewhat
7	definitely	not_definitely
		maybe
8	completely	not_completely
	absolutely	not_absolutely
		hardly
9	positively	not_positively
		vaguely
10	unbelievably	not_unbelievably

F R E Q U E N C Y M O D I F I E R S

Category	Modifier	"Not" Modifier
1	true	not_true
		false
2	frequently	not_frequently
		occasionally
3	often	not_often
		sometimes
4	rather_often	not_rather_often
	quite_often	not_quite_often
	usually	not_usually
5	very_often	not_very_often
	very_frequently	not_very_frequently
		almost_never
6	always	not_always
		never
7	definitely_always	definitely_never
8	absolutely_always	absolutely_never
9	positively_always	positively_never
10	unbelievably_always	unbelievably_never

Appendix C: Threshold Mapping Function

The toolkit contains threshold logic which compares the possibility distribution of a satisfying fact or rule with the possibility distribution of the clause to be resolved. If the distributions are within the threshold set by the user, the fact or rule satisfies the clause. In order to compare the possibility distributions, a rating scheme calculates a value for each distribution and the toolkit determines the difference. A negative difference implies the clause is more than satisfied (threshold does not apply). A positive result requires a threshold check. The difference in the possibility distributions is mapped to an acceptable threshold value using the function described below. If the user's threshold is greater than the acceptable threshold, the clause fails. Otherwise the fact or rule will satisfy the clause.

THRESHOLD MAPPING FUNCTION

Rating Difference	Acceptable Threshold
0 - 0.2	50
0.2 - 0.5	49
0.5 - 0.8	48
0.8 - 1.0	47
1.0 - 1.2	46
1.2 - 1.5	45
1.5 - 1.8	44
1.8 - 2.0	43
2.0 - 2.3	42
2.3 - 2.7	41
2.7 - 3.0	40
3.0 - 3.3	39
3.3 - 3.7	38
3.7 - 4.0	37
4.0 - 4.3	36
4.3 - 4.7	35
4.7 - 5.0	34
5.0 - 5.5	33
5.5 - 6.0	32
6.0 - 6.5	31
6.5 - 7.0	30
7.0 - 7.5	29
7.5 - 8.0	28
8.0 - 8.5	27
8.5 - 9.0	26
9.0 - 9.5	25
9.5 - 10.0	24
10 - 11	23
11 - 12	22
12 - 13	21
13 - 14	20
14 - 15	19
15 - 16	18
16 - 17	17
17 - 18	16
18 - 19	15
19 - 20	14
20 - 21	13
21 - 22	12
22 - 23	11
23 - 24	10
24 - 25	9
25 - 26	8
26 - 27	7
27 - 28	6
28 - 29	5
29 - 30	4

30 - 31	3
31 - 33	2
33 - 35	1
35 - 36	0

Appendix D: Output Generated by the Toolkit

The toolkit generates five types of output, each identified by a unique prefix:

1) Toolkit-InputN: input messages which prompt the user for data (the toolkit will wait for a response).
2) Toolkit-AuditN: audit messages which flag an error when the user has input incorrect data.
3) Toolkit-InfoN: information messages which inform the user of some result generated by the toolkit.
4) Toolkit-DiagN: diagnostic information produced by the toolkit as it executes an expert system. These messages are only produced if requested by the user.
5) Toolkit-ProbN: problem messages from the toolkit. In the unlikely event that one of these messages should appear, please contact the support person responsible for maintaining the toolkit. These messages are useful in debugging the toolkit.

The "N" suffix on each message identifier, is a number indicating the exact message within the toolkit. A complete list of the toolkit messages follows.

Toolkit-Input1: Enter first linguistic value
(enter done if none):

Toolkit-Input2: Enter next linguistic value
(enter done if none):

Toolkit-Input3: Enter domain value number x:

Toolkit-Input4: Enter degree of membership number x:

Toolkit-Input5: Enter linguistic value to review:

Toolkit-Input6: Enter modifier to review:

Toolkit-Input7: Enter next linguistic value
(enter s for same or d for done):

Toolkit-Input8: Display diagnostics as toolkit runs (y/n)?

Toolkit-Input9: Invoke thresholding in toolkit runs (y/n)?

Toolkit-Input10: Enter threshold value (0,50),
where 50 is the most stringent value:

Toolkit-Audit1: Domain value already exists.
Please reenter domain value number x:

Toolkit-Audit2: Degree of membership must be in range [0,1].
Please reenter degree of membership number x:

Toolkit-Audit3: Linquistic value: xxxxxx
does not exist with modifier: yyyyyy

Toolkit-Audit4: The toolkit does not recognize fuzzy modifier:
xxxxxx

Toolkit-Audit5: Linguistic value was not predefined: xxxxxx

Toolkit-Audit6: Threshold must be in the range [0,50].
Please reenter threshold value:

Toolkit-Info1: Linguistic value setup complete.

Toolkit-Info2:

Domain Value	Degree of Membership
x1	y1
x2	y2
x3	y3
x4	y4
x5	y5
x6	y6

```
Toolkit-Info3: Expert system setup complete.
Toolkit-Info4: Threshold logic fails
               on xxxxxx yyyyyy
Toolkit-Info5: Final degrees of membership:
               x1  x2  x3  x4  x5  x6
Toolkit-Info6: No degrees of membership produced.

Toolkit-Diag1: Defining fact xxxxxx
               and content modifier yyyyyy:
               x1  x2  x3  x4  x5  x6
               with context modifier zzzzzz:
               y1  y2  y3  y4  y5  y6
Toolkit-Diag2: Defining rule xxxxxx
               with context modifier yyyyyy:
               x1  x2  x3  x4  x5  x6
Toolkit-Diag3: Defining clause xxxxxx
               and content modifier yyyyyy:
               x1  x2  x3  x4  x5  x6
Toolkit-Diag4: Threshold not required
               since xxxxxx yyyyyy
               is more than satisfied.
Toolkit-Diag5: Threshold not required.
               An exact match is made.
Toolkit-Diag6: Theshold must be less than
               or equal to xx
Toolkit-Diag7: Intermediate degrees of membership:
               x1  x2  x3  x4  x5  x6
```

```
Toolkit-Diag8: Using the and operator, combine:
               x1  x2  x3  x4  x5  x6
               and:
               y1  y2  y3  y4  y5  y6
               with result:
               z1  z2  z3  z4  z5  z6

Toolkit-Prob1: Failed update_pd.
Toolkit-Prob2: Failed spread_tru.
Toolkit-Prob3: Threshold result not found in table: xx
```

Appendix E: The "Should_Attend_College" Example

Section 3.3 utilizes a small PROLOG knowledge base as an example in describing the toolkit environment. The knowledge base determines from a very limited set of rules and facts whether "Steve should attend college". This appendix includes the actual toolkit output when the should_attend_college example is executed under the toolkit. Section 3.3.6 references this appendix specifically and often as the toolkit execution logic is described. For completeness, the example knowledge base is included here as well.

```
should_attend_college(X) :-
  interested_in(X,Y):definitely,
  rich(X)--absolutely.
interested_in(steve,dentistry):sortof.
interested_in(steve,computer_science):very_much.
interested_in(steve,mechanics):definitely.
rich(steve):very--really.
```

```
run(should_attend_college(steve)).
Toolkit-Input10: Enter threshold value (0,50),
                 where 50 is the most stringent value:
|: 30.
Toolkit-Diag1: Defining fact interested_in
               and content modifier sortof:
               0.1     0.3     0.5     0.7     0.5     0.3
               and context modifier null:
               0.1     0.3     0.5     0.7     0.5     0.3
Toolkit-Diag3: Defining clause interested_in
               and content modifier definitely:
               0     0     0     0     0.2     0.4
Toolkit-Diag6: Threshold must be less than
               or equal to 23
Toolkit-Info4: Thresholding logic fails
               on definitely interested_in
Toolkit-Diag1: Defining fact interested_in
               and content modifier very_much:
               0     0     0     0.1     0.3     0.5
               and context modifier null:
               0     0     0     0.1     0.3     0.5
Toolkit-Diag3: Defining clause interested_in
               and content modifier definitely:
               0     0     0     0     0.2     0.4
Toolkit-Diag6: Threshold must be less than
               or equal to 41
Toolkit_Diag1: Defining fact rich
```

```
                and content modifier very:
                0      0      0      0.1      0.3      0.5
                and context modifier really:
                0      0      0      0      0.4      0.7
Toolkit-Diag3: Defining clause rich
                and content modifer null:
                0      0.1      0.3      0.5      0.6      0.8
Toolkit-Diag4: Threshold not required
                since null rich
                is more than satisfied.
Toolkit-Diag8: Using the and operator, combine:
                0      0      0      0.1      0.3      0.5
                and:
                0      0      0      0      0.4      0.7
                with result:
                0      0      0      0      0.3      0.5
Toolkit-Diag2: Defining rule should_attend_college
                with context modifier absolutely:
                0      0      0      0      0      0.8
Toolkit-Diag3: Defining clause should_attend_college
                and content modifier null:
                0      0.1      0.3      0.5      0.7      0.9
Toolkit-Diag4: Threshold not required
                since null should_attend_college
                is more than satified.
Toolkit-Diag1: Defining fact interested_in
                and content modifier definitely:
```

```
                0     0     0     0     0.2     0.4
                and context modifier null:
                0     0     0     0     0.2     0.4
Toolkit-Diag3:  Defining clause interested_in
                and content modifier definitely:
                0     0     0     0     0.2     0.4
Toolkit-Diag5:  Threshold not required.
                An exact match is made.
Toolkit_Diag1:  Defining fact rich
                and content modifier very:
                0     0     0     0.1     0.3     0.5
                and context modifier really:
                0     0     0     0     0.4     0.7
Toolkit-Diag3:  Defining clause rich
                and content modifer null:
                0     0.1     0.3     0.5     0.6     0.8
Toolkit-Diag4:  Threshold not required
                since null rich
                is more than satisfied.
Toolkit-Diag8:  Using the and operator, combine:
                0     0     0     0     0.2     0.4
                and:
                0     0     0     0     0.4     0.7
                with result:
                0     0     0     0     0.2     0.4
Toolkit-Diag2:  Defining rule should_attend_college
                with context modifier absolutely:
```

```
                0     0     0     0     0     0.7
Tookit-Diag3: Defining clause should_attend_college
              and content modifier null:
              0     0.1     0.3     0.5     0.7     0.9
Toolkit-Diag4: Threshold not required
               since null should_attend_college
               is more than satified.
Toolkit-Diag7: Intermediate degrees of membership:
               0     0     0     0     0     0.8
Toolkit-Diag7: Intermediate degrees of membership:
               0     0     0     0     0     0.7
Toolkit-Diag5: Final degrees of membership:
               0     0     0     0     0     0.8
yes
| ?-
```

Appendix F: A Selection of Toolkit Program Listings

Four of the ten files which comprise the toolkit C-PROLOG code are contained in this final appendix. The files include:

- expert - main inferencing scheme logic driving the execution of the expert system
- thresh - contains the thresholding logic
- context - contains data context uncertainty logic
- truth - predefined truth tables and fuzzy modifiers

These files make up the core of the toolkit; the remaining files are trivial in comparison and are not included.

```
/* ------------ E X P E R T --------------- */
/*                                           */
/*  M. Bicker                       4/20/87  */
/*                                           */
/*  Main toolkit processiong logic. Applies  */
/*  toolkit inferencing scheme to expert     */
/*  system as is executes.                   */
/*                                           */
/* ----------------------------------------- */

gethis.

run(Exp_sys) :-
  setup, !,
  setof(Modf,sub_run(Exp_sys,Modf),Mlist),
  wrapup(Mlist).



/* Trun was set up to use for testing,  */
/* since the setof clause had the nasty */
/* effect of causing spy to begin at    */
/* the sub_run rule.                    */


trun(Exp_sys) :-
  setup, !,
  sub_run(Exp_sys,Modf),
  retract(threshold(Thresh)).


/* Sub_run executes the expert system for each  */
/* or clause. It pops any query term modifier   */
/* and then looks for the input query on the    */
/* expert system database. If found, it tries   */
/* to satisfy the body of the expert system.    */
/* From there it applies any context modifier   */
/* in rule_combine and checks the result with   */
/* the input query and the term modifier it     */
/* popped. It stores the result in fin_ling.    */


sub_run(Exp_sys,Modf) :-
  pop_mod(Exp_sys,TerMod,Real_ES),
  exp(Real_ES,Body,FactMod,TruMod),
  satisfy_body(Body,X1,X2,X3,X4,X5,X6),
  functor(Real_ES,Oper,Arity),
  rule_combine(Body,Oper,FactMod,TruMod,
               X1,X2,X3,X4,X5,X6,
               Y1,Y2,Y3,Y4,Y5,Y6),
  ck_thres(Oper,TerMod,Y1,Y2,Y3,Y4,Y5,Y6,Fail_Sw),
  increment_reg(final),
  final(Modf),
```

```
  assertz(fin_ling(Modf,Y1,Y2,Y3,Y4,Y5,Y6)),
  ck_fail(Fail_Sw).


/* Satisfy body controls the execution of the   */
/* expert system to the greatest extent. It     */
/* handles a couple of special cases (like a    */
/* null body) but then pulls off the operator   */
/* to check for a fact or a structure (a rule). */


satisfy_body(null,1,1,1,1,1,1).

satisfy_body(Body,X1,X2,X3,X4,X5,X6) :-
  not(Body==null),
  fail_thres(X),
  X==1.

satisfy_body(Body,X1,X2,X3,X4,X5,X6) :-
  not(Body==null),
  functor(Body,Oper,Arity),
  fact_or_struct(Oper,Body,X1,X2,X3,X4,X5,X6).


/* Fact_or_structure case 1: a rule is being   */
/* processed. Take the first clause and pop    */
/* off any term modifier (data content) and    */
/* check for a not situation (processes the    */
/* clause). Check the threshold of the result  */
/* with the query and the term modifier pop-   */
/* ped. Then satisfy the rest of the rule      */
/* clauses using satisfy_body. Combine the     */
/* results using the and operator when done.   */


fact_or_struct((','),Body,X1,X2,X3,X4,X5,X6) :-
  Body=..[Oper,Head,RestBody],
  pop_mod(Head,TerMod,RealHead),
  functor(RealHead,NewOper,Arity), !,
  check_for_not(NewOper,RealHead,Y1,Y2,Y3,Y4,Y5,Y6),
  ck_thres(NewOper,TerMod,Y1,Y2,Y3,Y4,Y5,Y6,Fail_Sw),
  set_fail(Fail_sw),
  satisfy_body(RestBody,Z1,Z2,Z3,Z4,Z5,Z6),
  and_combine(Y1,Y2,Y3,Y4,Y5,Y6,Z1,Z2,Z3,Z4,Z5,Z6,
              X1,X2,X3,X4,X5,X6,NewOper,RealHead),
  ck_fail(Fail_Sw).


/* Fact_or_Structure case 2: a fact is being  */
/* processed with a data content fuzzy modif- */
/* ier (or term mod). This is extracted and   */
/* the fact is processed by check_for_not.    */
/* The threshold is checked using the results,*/
```

```
/* the query and the term modifier removed.   */

fact_or_struct((':'),Body,X1,X2,X3,X4,X5,X6) :-
  Body=..[Oper,Fact,TerMod],
  functor(Fact,NewOper,Arity),
  check_for_not(NewOper,Fact,X1,X2,X3,X4,X5,X6),
  ck_thres(NewOper,TerMod,X1,X2,X3,X4,X5,X6,Fail_Sw),
  ck_fail(Fail_Sw).



/* Fact_or_structure case 3: an unmodifed fact is  */
/* being processed. Use check_for_not to get the   */
/* result and check the threshold with an unmodif- */
/* ied query.                                      */



fact_or_struct(Oper,Body,X1,X2,X3,X4,X5,X6) :-
  not(Oper==','),
  not(Oper==':'),
  check_for_not(Oper,Body,X1,X2,X3,X4,X5,X6),
  ck_thres(Oper,null,X1,X2,X3,X4,X5,X6,Fail_Sw),
  ck_fail(Fail_Sw).



/* check_for_not either finds the fnot operator  */
/* or it does not. In the former case, it uses   */
/* the not of the result of check_builtin as the */
/* result. In the latter case, it just calls     */
/* check_builtin.                                */


check_for_not(fnot,Head,1,1,1,1,1,1) :-
  arg(1,Head,NewHead), !,
  functor(NewHead,Oper,Arity),
  not(check_builtin(Oper,NewHead,X1,X2,X3,X4,X5,X6)).

check_for_not(Oper,Head,X1,X2,X3,X4,X5,X6) :-
  not(Oper==fnot), !,
  check_builtin(Oper,Head,X1,X2,X3,X4,X5,X6).


/* check_builtin is essentially looking for   */
/* builtin Prolog functions and executing     */
/* them as is when they are found. The tool-  */
/* kit contains only a subset of all avail-   */
/* able prolog functions, but more could be   */
/* easily added. If the function is not a     */
/* builtin function (including the cut), then */
/* the toolkit must check the expert system   */
/* database for the clause in chk_exp and     */
```

```
/* execute the new clause using satisfy_body. */
/* Rule_combine applies any data context      */
/* fuzzy modifer.                             */


check_builtin(stamp,Head,1,1,1,1,1,1) :-
  gethis.

check_builtin(var,Head,1,1,1,1,1,1) :-
  call(Head).

check_builtin(write,Head,1,1,1,1,1,1) :-
  call(Head).

check_builtin(read,Head,1,1,1,1,1,1) :-
  call(Head).

check_builtin(!,Head,1,1,1,1,1,1).

check_builtin(!,Head,1,1,1,1,1,1) :-
  set_reg(cuton,1), !,
  fail.

check_builtin(fail,Head,1,1,1,1,1,1) :-
  !, fail.

check_builtin(Oper,Head,X1,X2,X3,X4,X5,X6) :-
  set_reg(cuton,0), !,
  chk_exp(Head,NewBody,FactMod,TruMod),
  satisfy_body(NewBody,Y1,Y2,Y3,Y4,Y5,Y6),
  rule_combine(NewBody,Oper,FactMod,TruMod,
               Y1,Y2,Y3,Y4,Y5,Y6,
               X1,X2,X3,X4,X5,X6).


/* chk_exp decides whether a cut has occurred or */
/* not to control rollbacks and backtracking.    */


chk_exp(Head,NewBody,FactMod,TruMod) :-
  cuton(X), !,
  X==0,
  exp(Head,NewBody,FactMod,TruMod).

chk_exp(Head,NewBody,FactMod,TruMod) :-
  exp(Head,NewBody,FactMod,TruMod).
  cuton(X),
  X==1, !,
  fail.

pop_mod(Head,TerMod,RealHead) :-
  Head = (RealHead:TerMod), !.
```

```
pop_mod(Head,TerMod,RealHead) :- !.


set_fail(1) :-
  set_reg(fail_thres,1), !.

set_fail(0) :-
  set_reg(fail_thres,0), !.


ck_fail(1) :-
  fail, !.

ck_fail(0) :- !.


/* Rule combine case 1: fact or rule without */
/* a lingval. Return same value.             */


rule_combine(Body,Oper,FactMod,TruMod,
             X1,X2,X3,X4,X5,X6,
             X1,X2,X3,X4,X5,X6) :-
  not(lingval(Oper,Mod,Dom,Deg)), !.


/* Rule combine case 2: a fact; determine  */
/* lingval with Oper and FactMod, then     */
/* apply TruMod to the result. In this     */
/* case the X values are meaningless,      */
/* since there is no body.                 */


rule_combine(null,Oper,FactMod,TruMod,
             X1,X2,X3,X4,X5,X6,
             Y1,Y2,Y3,Y4,Y5,Y6) :-
  ck_diag2(Oper,FactMod),
  set_temps(Oper,FactMod),
  ck_diag_tab,
  get_ling(Z1),
  get_ling(Z2),
  get_ling(Z3),
  get_ling(Z4),
  get_ling(Z5),
  get_ling(Z6),
  ck_diag3(TruMod),
  apply_tru(TruMod,Z1,Z2,Z3,Z4,Z5,Z6,
            Y1,Y2,Y3,Y4,Y5,Y6), !.


/* Rule combine case 3: a rule; FactMod   */
/* does not apply. Apply TruMod to the    */
/* results (the Xs) of satisfying the     */
```

```
/* body.                                          */

rule_combine(Body,Oper,FactMod,TruMod,
             X1,X2,X3,X4,X5,X6,
             Y1,Y2,Y3,Y4,Y5,Y6) :-
  ck_diag4(Oper,TruMod),
  apply_tru(TruMod,X1,X2,X3,X4,X5,X6,
            Y1,Y2,Y3,Y4,Y5,Y6), !.


and_combine(X1,X2,X3,X4,X5,X6,Y1,Y2,Y3,Y4,Y5,Y6,
            Z1,Z2,Z3,Z4,Z5,Z6,NewOper,Head) :-
  fail_thres(X),
  X==1.

and_combine(X1,X2,X3,X4,X5,X6,Y1,Y2,Y3,Y4,Y5,Y6,
            Z1,Z2,Z3,Z4,Z5,Z6,NewOper,Head) :-
  set_reg(defpd,0),
  ck_diag9(X1,X2,X3,X4,X5,X6,
           Y1,Y2,Y3,Y4,Y5,Y6),
  ck_def_pd(X1,X2,X3,X4,X5,X6,Y1,Y2,Y3,Y4,Y5,Y6,
            Z1,Z2,Z3,Z4,Z5,Z6), !.


/* apply_tru will apply the data context modifier  */
/* if one exists (by calling context). If one does */
/* not exist, it still prints out the values.      */


apply_tru(null,X1,X2,X3,X4,X5,X6,X1,X2,X3,X4,X5,X6) :-
  ck_diag_tab,
  ck_diag1(X1),
  ck_diag1(X2),
  ck_diag1(X3),
  ck_diag1(X4),
  ck_diag1(X5),
  ck_diag1(X6),
  ck_diag_nl.


apply_tru(TruMod,X1,X2,X3,X4,X5,X6,Y1,Y2,Y3,Y4,Y5,Y6) :-
  context(X1,X2,X3,X4,X5,X6,Y1,Y2,Y3,Y4,Y5,Y6,TruMod),
  ck_diag_tab,
  ck_diag1(Y1),
  ck_diag1(Y2),
  ck_diag1(Y3),
  ck_diag1(Y4),
  ck_diag1(Y5),
  ck_diag1(Y6),
  ck_diag_nl.


set_temps(Oper,Mod) :-
```

```
  lingval(Oper,Mod,Dom,Deg),
  assertz(temp_ling(Deg)),
  fail.

set_temps(Oper,Mod).

get_ling(X1) :-
  temp_ling(X1),
  ck_diag1(X1),
  retract(temp_ling(X1)).

ck_def_pd(X1,X2,X3,X4,X5,X6,Y1,Y2,Y3,Y4,Y5,Y6,
          Z1,Z2,Z3,Z4,Z5,Z6) :-
  defpd(X),
  X==0,
  ck_diag_tab,
  min_ling(X1,Y1,Z1),
  min_ling(X2,Y2,Z2),
  min_ling(X3,Y3,Z3),
  min_ling(X4,Y4,Z4),
  min_ling(X5,Y5,Z5),
  min_ling(X6,Y6,Z6),
  ck_diag_nl.

ck_def_pd(X1,X2,X3,X4,X5,X6,Y1,Y2,Y3,Y4,Y5,Y6,
          Z1,Z2,Z3,Z4,Z5,Z6) :-
  min(Z1,X1,Y1),
  min(Z2,X2,Y2),
  min(Z3,X3,Y3),
  min(Z4,X4,Y4),
  min(Z5,X5,Y5),
  min(Z6,X6,Y6).


min_ling(X1,Y1,Z1) :-
  min(Z1,X1,Y1),
  ck_diag1(Z1).
```

```
/* --------------- T H R E S H --------------- */
/*                                             */
/*  M. Bicker                        4/20/87   */
/*                                             */
/*  Thresh controls all the thresholding logic */
/*  in the toolkit. It includes the threshold  */
/*  mapping function at the end of the file.   */
/*                                             */
/* ------------------------------------------- */

/* ck_thresh inputs the query operator and any   */
/* data content modifier (TerMod). It calculates */
/* a value from this input. It also inputs six   */
/* values which are degrees of membership        */
/* of the result from executing the query. A     */
/* second value is computed from these six input */
/* numbers. The two values are then compared.    */
/* If the difference is less than the threshold  */
/* an error is reported.                         */

ck_thres(Oper,TerMod,X1,X2,X3,X4,X5,X6,0) :-
  get_val1(Oper,TerMod,Val1),
  add_val(X1,X2,X3,X4,X5,X6,Val2),
  comp_vals(Val1,Val2,Oper,TerMod), !.

ck_thres(Oper,TerMod,X1,X2,X3,X4,X5,X6,1) :-
  nl,
  write('Toolkit-Info4: Thresholding logic fails '),
  nl,
  tab(15), write('on '),
  write(TerMod),
  write(' '),
  write(Oper), nl, !,
  fail.

/* get-val1 audits the operator on the lingval */
/* database (is it a valid linguistic value?). */
/* If it is not defined and no TerMod is being */
/* applied, then val1 will be set to 100 such  */
/* that no threshold checking is done. This is */
/* the case for most PROLOG builting functions.*/
/* Once the operator is audited, calc_val is   */
/* called. Note that audit_lingval is in the   */
/* lingset file.                               */


get_val1(Oper,null,Val1) :-
  lingval(Oper,null,Dom,Deg),
  calc_val(Oper,null,Val1), !.

get_val1(Oper,null,100) :-
```

```
get_val1(Oper,TerMod,Val1) :-
  audit_lingval(Oper,TerMod), !,
  calc_val(Oper,TerMod,Val1), !.


/* calc_val just sets up the values for add_val */


calc_val(Oper<terMod,Val1) :-
  ck)diag5(Oper,TerMod),
  set_temps(Oper,TerMod),
  ck_diag_tab,
  get_temp(X1),
  get_temp(X2),
  get_temp(X3),
  get_temp(X4),
  get_temp(X5),
  get_temp(X6),
  ck_diag_nl,
add_val(X1,X2,X3,X4,X5,X6,Val1).

get_temp(X) :-
  temp_ling(X),
  retract(temp_ling(X)),
  ck_diag1(X).


/* add_val incorporates the rating scheme   */
/* used to interpret six degrees of memb-   */
/* ership. Each value is interegated for    */
/* a 1, 0 or other. Rating points are       */
/* assigned by get_points and all are       */
/* added together by add_points.            */



add_val(X1,X2,X3,X4,X5,X6,Val) :-
  get_points(X1,P1),
  get_points(X2,P2),
  get_points(X3,P3),
  get_points(X4,P4),
  get_points(X5,P5),
  get_points(X6,P6),
  add_points(X1,X6,P1,P2,P3,P4,P5,P6,Val), !.

get_points(0,3).
get_points(1,2).
get_points(X,X).


add_points(X1,X6,P1,P2,P3,P4,P5,P6,NewVal) :-
  Val is P1 + P2 + P3 + P4 + P5 + P6,
  chk_neg(X1,X6,Val,NewVal), !.
```

```
/* chk_neg looks for the peak of the  */
/* fuzzy subset. If it is to the left */
/* the result is negated.             */


chk_neg(X1,X6,Val,NewVal) :-
  X1 > X6,
  NewVal is Val * -1.

chk_neg(X1,X6,Val,Val).


/*  comp_vals checks for val1 = 100 to    */
/* exclude builtin functions from the     */
/* threshold checking. Since add_points   */
/* returns a number between -18 and 18,   */
/* both values are normalized to the      */
/* range 0 to 236. The difference between */
/* the two numbers (Val1 and Val2) can    */
/* be between -36 and 36. Convert is      */
/* called to map the difference to a      */
/* threshold value and this is then       */
/* compared to the user's threshold in    */
/* aud_thres.                             */


comp_vals(100,Val2,Oper(TerMod).

comp_vals(Val1,Val2,Oper(TerMod) :-
  Temp1 is Val1 + 18,
  Temp2 is Val2 + 18,
  Temp is Temp1 - Temp2,
  convert(Temp,ConvTemp,Oper,TerMod),
  threshold(Thresh), !,
  aud_thres(ConvTemp,Thresh).



/*  convert changes any negative number to an */
/* acceptable threshold of 51, because the    */
/* user's threshold is always less and will   */
/* pass. The satisfying clause more than      */
/* satisfies the query (a negative difference)*/
/* If the number is not negative, convert     */
/* searches through the threshold mapping     */
/* table for a match. It retrieves the match- */
/* ing threshold and returns. An error occurs */
/* when the table does not produce a match.   */
```

```
convert(Temp,ConvTemp,Oper,TerMod) :-
  Temp < 0,
  ConvTemp is 51,
  ck_diag6(Oper,TerMod).

convert(Temp,ConvTemp,Oper,TerMod) :-
  table(Bot,Top,ConvTemp),
  Temp >= Bot,
  Temp =< Top,
  ck_diag7(ConvTemp).

convert(Temp,0,Oper,TerMod) :-
  nl,
  write('Toolkit-Prob3: Threshold result not found in table: '),
  write(Temp), nl.


/* aud)thresh simply ensures that the user's */
/* threshold is less than calculated thresh- */
/* old. Otherwise, it fails.                 */


aud_thres(Temp,Thresh) :-
  Temp >= Thresh.

aud_thres(Temp,Thresh) :-
  fail.


/* the threshold mapping function follows */


table(0,0,2,50).
table(0.2,0.5,49).
table(0.5,0.8,48).
table(0.8,1.0,47).
table(1.0,1.2,46).
table(1.2,1.5,45).
table(1.5,1.8,44).
table(1.8,2.0,43).
table(2.0,2.3,42.
table(2.3,2.7,41).
table(2.7,3.0,40).
table(3.0,3.3,39).
table(3.3,3.7,38).
table(3.7,4.0,37).
table(4.0,4.3,36).
table(4.3,4.7,35).
table(4.7,5.0,34).
table(5.0,5.5,33).
table(5.5,6.0,32).
table(6.0,6.5,31).
table(6.5,7.0,30).
```

```
table(7.0,7.5,29).
table(7.5,8.0,28).
table(8.0,8.5,27).
table(8.5,9.0,26).
table(9.0,9.5,25).
table(9.5,10.0,24).
table(10,11,23).
table(11,12,22).
table(12,13,21).
table(13,14,20).
table(14,15,19).
table(15,16,18).
table(16,17,17).
table(17,18,16).
table(18,19,15).
table(19,20,14).
table(20,21,13).
table(21,22,12).
table(22,23,10).
table(24,25,9).
table(25,26,8).
table(26,27,7).
table(27,28,6).
table(28,29,5).
table(29,30,4).
table(30,31,3).
table(31,33,2).
table(33,35,1).
table(35,36,0).
```

```
/* -------------- C O N T E X T ------------- */
/*                                            */
/*  M. Bicker                      4/20/87    */
/*                                            */
/*  Context applies a data context uncertainty */
/*  modifier to a fuzzy subset.               */
/*                                            */
/* ------------------------------------------ */

/* Context inputs six degrees of membership and  */
/* a data context modifier (TruMod). The         */
/* output is six modified degrees of membership. */
/* Context determines the fuzzy subset peak and  */
/* calls update_pd.                              */

context(X1,X2,X3,X4,X5,X6,Y1,Y2,Y3,Y4,Y5,Y6,
        TruMod) :-
  X1 < X6,
  update_pd(X1,X2,X3,X4,X5,X6,Y1,Y2,Y3,Y4,Y5,Y6,
            TruMod).

context(X1,X2,X3,X4,X5,X6,Y1,Y2,Y3,Y4,Y5,Y6,
        TruMod) :-
  update_pd(X6,X5,X4,X3,X2,X1,Y6,Y5,Y4,Y3,Y2,Y1,
            TruMod).



/* update_pd retrieves the data context fuzzy    */
/* modifier definition from the truth tables     */
/* and determines the strength of the modifier.  */
/* It then modifies the possibility distribution */
/* by calling spread_tru.                        */



update_pd(A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6,
          TruMod) :-
  lingval(true,TruMod,_,_),
  set_temps(true,TruMod),
  add_trumod(Value),
  truedef(TruMod,NotInd,Numb),
  spread_tru(A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6,
             NotInd,Value).

update_pd(A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6,
          TruMod) :-
  write('Toolkit-Prob1: Failed update_pd'), nl.


add_trumod(Value) :-
  temp_ling(D1),
```

```
  retract(temp_ling(D1)),
  add_trumod(Temp),
  Value is Temp + D1.

add_trumod(0).


/* spread_tru determines how strong the modifier is */
/* by checking the calculated values. There are 3   */
/* positive modifier categories and 3 negative mod- */
/* ifier categories. Check-zero is called with 3    */
/* modifying values depending on data context mod.  */


spread_tru(A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6,
           nott,Value) :-
  Value < 4,
  check_zero(A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6,
             0,0,0.1).

spread_tru(A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6,
           nott,Value) :-
  Value < 5.1,
  check_zero(A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6,
             0,0.1,0.2).

spread_tru(A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6,
           nott,Value) :-
  check_zero(A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6,
             0.1,0.2,0.3).

spread_tru(A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6,
           null,Value) :-
  Value > 2,
  check_zero(A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6,
             0,0,0.1).

spread_tru(A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6,
           null,Value) :-
  Value > 0.9,
  check_zero(A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6,
             0,0.1,0.2).

spread_tru(A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6,
           null,Value) :-
  check_zero(A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6,
             0.1,0.2,0.3).

spread_tru(A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6,
           NotInd,Value) :-
  write('Toolkit-Prob2: Failed spread_tru '), nl.
```

```
/* check zero determines where the boundary edge */
/* is by searching for the first non-zero degree */
/* of membership. Note that negative fuzzy sub-  */
/* sets are reversed (see context above) so that */
/* this routine works for both cases (pos & neg).*/
/* Check_zero then increases and reduces degrees */
/* of membership using the modifying values input*/


check_zero(A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6,
           Val1,Val2,Val3) :-
  not(A1==0),
  reduce(A1,B1,Val3),
  reduce(A2,B2,Val2),
  reduce(A3,B3,Val1),
  increase(A4,B4,Val1),
  increase(A5,B5,Val2),
  increase(A6,B6,Val3).

check_zero(A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6,
           Val1,Val2,Val3) :-
  not(A2==0),
  reduce(A2,B2,Val3),
  reduce(A3,B3,Val2),
  increase(A4,B4,Val1),
  increase(A5,B5,Val2),
  increase(A6,B6,Val3).

check_zero(A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6,
           Val1,Val2,Val3) :-
  not(A3==0),
  reduce(A3,B3,Val3),
  reduce(A4,B4,Val2),
  increase(A5,B5,Val2),
  increase(A6,B6,Val3).

check_zero(A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6,
           Val1,Val2,Val3) :-
  not(A4==0),
  reduce(A4,B4,Val3),
  increase(A5,B5,Val2),
  increase(A6,B6,Val3).

check_zero(A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6,
           Val1,Val2,Val3) :-
  reduce(A5,B5,Val3),
  increase(A6,B6,Val3).


increase(A,B,Increm) :-
  Temp is A + Increm,
  chk_pos_result(Temp,B).
```

```
reduce(A,B,Increm) :-
  Temp is A - Increm,
  chk_neg_result(A,Temp,B).

chk_pos_result(Temp,1) :-
  Temp > 1.

chk_pos_result(Temp,Temp).

chk_neg_result(A,0,0).

chk_neg_result(A,Temp,0) :-
  Temp < 0.

/* reduce to zero or don't reduce at all */

chk_neg_result(A,Temp,A).
```

```
/* --------------- T R U T H ----------------- */
/*                                             */
/*  M. Bicker                        4/20/87   */
/*                                             */
/*  Truth contains the predefined truth        */
/*  tables and valid fuzzy modifiers given     */
/*  by the toolkit.                            */
/*                                             */
/* ------------------------------------------- */

/* truedef defines all the fuzzy modifiers by */
/* specifying the modifier name, an indicator */
/* of whether it is a positive (null) or neg- */
/* ative modifier (nott) and the category of  */
/* the truth tables which describes the mod.  */


truedef(unbelievably,null,10).
truedef(positively,null,9).
truedef(completely,null,8).
truedef(absolutely,null,8).
truedef(definitely,null,7).
truedef(very,null,6).
truedef(very_much,null,6).
truedef(very_likely,null,6).
truedef(alot,null,5).
truedef(basically,null,5).
truedef(quite,null,4).
truedef(quite_abit,null,4).
truedef(generally,null,4).
truedef(really,null,3).
truedef(fairly,null,3).
truedef(truely,null,2).
truedef(mostly,null,2).
truedef(true,null,1).

truedef(almost,nott,2).
truedef(kindof,nott,2).
truedef(sortof,nott,2).
truedef(a_little,nott,2).
truedef(somewhat,nott,2).
truedef(maybe,nott,2).
truedef(hardly,nott,2).
truedef(vaguely,nott,2).

truedef(not_unbelievably,nott,10).
truedef(not_positively,nott,9).
truedef(not_completely,nott,8).
truedef(not_absolutely,nott,8).
truedef(not_definitely,nott,7).
truedef(not_very,nott,6).
```

```
truedef(not_very_much,nott,6).
truedef(not_alot,nott,5).
truedef(not_quite,nott,4).
truedef(not_really,nott,3).
truedef(not_truely,nott,2).
truedef(not_true,nott,1).
truedef(false,nott,1).

truedef(unbelievably_always,null,10).
truedef(positively_always,null,9).
truedef(absolutley_always,null,8).
truedef(definitely_always,null,7).
truedef(always,null,6).
truedef(very_often,null,5).
truedef(very_frequently,null,5).
truedef(usually,null,4).
truedef(rather_often,null,4).
truedef(quite_often,null,4).
truedef(often,null,3).
truedef(frequently,null,2).

truedef(unbelievably_never,nott,10).
truedef(positively_never,nott,9).
truedef(absolutley_never,nott,8).
truedef(definitely_never,nott,7).
truedef(never,nott,6).
truedef(not_always,nott,6).
truedef(almost_never,nott,5).
truedef(not_very_often,nott,5).
truedef(not_very_frequently,nott,5).
truedef(not_usually,nott,4).
truedef(not_rather_often,nott,4).
truedef(not_quite_often,nott,4).
truedef(sometimes,nott,3).
truedef(not_often,nott,3).
truedef(occasionally,nott,2).
truedef(not_frequently,nott,2).


/* the toolkit defines the truth tables in terms  */
/* of 11 categories of truth. The lingval fact    */
/* is used with the linguistic value of true and  */
/* the modifier equal to the category number.     */
/* The toolkit uses the lingval database to store */
/* all possibility distributions.                 */


lingval(true,11,1,0).
lingval(true,11,2,0).
lingval(true,11,3,0).
lingval(true,11,4,0).
lingval(true,11,5,0).
lingval(true,11,6,0).
```

```
lingval(true,10,1,0).
lingval(true,10,2,0).
lingval(true,10,3,0).
lingval(true,10,4,0).
lingval(true,10,5,0).
lingval(true,10,6,0.1).

lingval(true,9,1,0).
lingval(true,9,2,0).
lingval(true,9,3,0).
lingval(true,9,4,0).
lingval(true,9,5,0).
lingval(true,9,6,0.2).

lingval(true,8,1,0).
lingval(true,8,2,0).
lingval(true,8,3,0).
lingval(true,8,4,0).
lingval(true,8,5,0.1).
lingval(true,8,6,0.3).

lingval(true,7,1,0).
lingval(true,7,2,0).
lingval(true,7,3,0).
lingval(true,7,4,0).
lingval(true,7,5,0.2).
lingval(true,7,6,0.4).

lingval(true,6,1,0).
lingval(true,6,2,0).
lingval(true,6,3,0).
lingval(true,6,4,0.1).
lingval(true,6,5,0.3).
lingval(true,6,6,0.5).

lingval(true,5,1,0).
lingval(true,5,2,0).
lingval(true,5,3,0).
lingval(true,5,4,0.2).
lingval(true,5,5,0.4).
lingval(true,5,6,0.6).

lingval(true,3,1,0).
lingval(true,3,2,0).
lingval(true,3,3,0.2).
lingval(true,3,4,0.4).
lingval(true,3,5,0.6).
lingval(true,3,6,0.8).

lingval(true,2,1,0).
lingval(true,2,2,0.1).
lingval(true,2,3,0.3).
```

```
lingval(true,2,4,0.5).
lingval(true,2,5,0.7).
lingval(true,2,6,0.9).

lingval(true,1,1,0).
lingval(true,1,2,0.2).
lingval(true,1,3,0.4).
lingval(true,1,4,0.6).
lingval(true,1,5,0.8).
lingval(true,1,6,1).

lingval(true,null,1,0).
lingval(true,null,2,0.2).
lingval(true,null,3,0.4).
lingval(true,null,4,0.6).
lingval(true,null,5,0.8).
lingval(true,null,6,1).
```