

Inexact Reasoning In Prolog-Based Expert Systems

Koenraad G. Lecot

**January 1986
CSD-860053**

UNIVERSITY OF CALIFORNIA

Los Angeles

**Inexact Reasoning in
Prolog-Based Expert Systems**

A thesis submitted in partial satisfaction of the
requirement for the degree Master of Science
in Computer Science

by

Koenraad Georgius Lecot

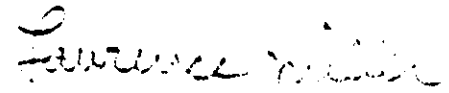
1984

© Copyright by
Koenraad Georgius Lecot
1984

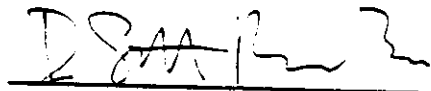
The thesis of Koenraad Georgius Lecot is approved.



Michael Dyer



Lawrence Miller



D. Stott Parker, Committee Chair

University of California, Los Angeles

1984

TABLE OF CONTENTS

	page
1. Introduction	1
1.1 Expert Systems	1
1.2 Prolog for Expert Systems	2
1.3 Overview of the Thesis	4
2. The Prolog Programming Language	7
2.1 Introduction	7
2.2 The Horn Clause Subset of Logic	8
2.3 Prolog and Expert Systems	14
3. Inexact Reasoning in Expert Systems	16
3.1 Introduction	16
3.2 Inexact Reasoning	19
3.3 Mycin	22
3.4 Prospector	25
3.5 The Dempster-Schafer Theory	27
3.6 Fuzzy Logic	29
4. Inexact Reasoning in Prolog	33
4.1 Introduction	33
4.2 Probabilistic Reasoning	34
4.3 Fundamentals of Probabilistic Reasoning in Prolog	35
4.4 Interpreters for Logic Programs with Uncertainties	37

5. Subjective Bayesian Strategy	41
5.1 Introduction	41
5.2 Uncertainty of Evidence	43
5.3 Multiple Evidence	44
5.4 Conclusion	46
5.5 Problems with Shapiro's Method	47
5.6 Prolog Implementation of Prospector	50
6. Belief Strategy (Mycin)	57
6.1 Introduction	57
6.2 The Consultation Subsystem	58
6.3 Explanations	62
6.4 Prolog Implementation of Mycin	62
6.5 Conclusion	76
7. Comparison of Probabilistic Approaches to Inexact Reasoning	77
7.1 Introduction	77
7.2 Knowledge Representation of Uncertainty	78
7.3 Assumptions in the Inference	80
7.4 The Control Structure for Propagating Uncertainty	82
7.5 Dealing with Inconsistent Information	82
8. Fuzzy Logic	83
8.1 Introduction	83
8.2 Fuzzy Set Theory	84
8.3 The Concept of a Linguistic Variable	91
8.4 Prolog Implementation of Fuzzy Sets	95

8.5 Fuzzy Databases	95
8.6 Fuzzy Inference	105
8.7 Psychological Considerations of Fuzziness	109
8.8 Conclusion	110
9. Conclusion and Future Work	112
9.1 Motivation for this Work	112
9.2 Conclusion	114
9.3 Future Work	116
Bibliography	119
Appendices	127
1. Prolog Implementation of Prospector	128
2. Execution Trace of Prospector in Prolog	141
3. Prolog Implementation of Mycin	148
4. Execution Trace of Mycin in Prolog	164
5. Prolog Implementation of Fuzzy Set Operators	168
6. Prolog Implementation of a Fuzzy Database	173

ACKNOWLEDGEMENTS

It is my pleasure to acknowledge the help and contributions of the following people and organizations and offer them my sincere gratitude:

Stott Parker, advisor and friend, who told me about Prolog from the moment I arrived at UCLA. I would like to thank him for his help and advice, both during this thesis project and during my stay in UCLA's computer science department.

Russ Abbott, Kamran Parsaye and Tulin Mangir. Their critical reviews of earlier drafts of this thesis are much appreciated.

Larry Miller and Mike Dyer, for serving on my committee and for their useful remarks on this thesis. I especially like to thank Mike as this project started under his direction.

Rik Verstraete, for his critical reading of this work, his useful suggestions and most of all, for his help with the formatting of this text.

Safaa Hashim, with whom I had enlightening interactions on fuzzy logic, and Christian Valcke for proofreading this manuscript.

Finally, the Belgian American Educational Foundation, for financial support during my first year at UCLA, and the IBM corporation, who jointly funded a University of California MICRO grant under which part of this research was conducted.

ABSTRACT OF THE THESIS

Inexact Reasoning in Prolog-Based Expert Systems

by

Koenraad Georgius Lecot

Master of Science in Computer Science
University of California, Los Angeles, 1984
Professor D. Stott Parker, Chair

Expert systems are only worthy their name if they can cope in a consistent and natural way with the uncertainty and vagueness that is inherent to real world expertise. This thesis explores the current methodologies, both in the light of their acceptability and of their implementation in the logic programming language Prolog. We treat in depth the subjective Bayesian approach to inexact reasoning and describe a meta-level implementation in Prolog. This probabilistic method is compared with an alternative theory of belief used in Mycin. We describe an implementation of Mycin's consultation phase. We argue further that the theory of fuzzy logic is more adequate to describe the uncertainty and vagueness of real world situations. Fuzzy logic is put in contrast with the probabilistic approaches and an implementation strategy is described.

CHAPTER 1

INTRODUCTION

1.1. Expert Systems

An *expert system* is a software system whose behavior in a given problem solving area is close to that of a human being with expertise in the given domain. Expert systems have been designed for many domains including medical diagnosis, geology, analysis of chemical structures and computer configuration. New applications arise as computing power becomes cheaper and more generally available.

Expert systems use many of the techniques of *Artificial Intelligence (AI)*, a field where the purely deterministic execution of a program is replaced by an apparently spontaneous reasoning. Human knowledge is an essential part of all expert systems and is stored as the *knowledge base*. The expert system interprets this knowledge, performing the logical deductions and inspiring the guesses and decisions that might otherwise have been performed by the human expert himself.

Expert systems function as computerized inference engines. They deduce relationships from a collection of *rules* which are also part of the knowledge base. In traditional expert systems, such as Mycin,⁹⁴ the inference rules are of the general format:

IF <symptom-1> *AND* <symptom-2> *AND* ... *AND* <symptom-n>

THEN <conclusion>

The rules are usually nested so that the conclusion of one may become a symptom or cause of another. When trying to prove a conclusion, the expert system uses deductive inference to generate a virtual search tree by backward-chaining through the rules. From this tree, a solution may eventually be found when facts,

i.e. rules with no descendants, are reached. In order to express uncertainty about rules one may associate a certainty measure with their preconditions enabling their conclusions to be given with a specified degree of certainty.

An *expert system*, as its name implies, is an information system which provides the user with a facility for posing questions and obtaining answers related to the expert knowledge stored in its knowledge base. Typically, such systems possess a nontrivial inferential capability and, in particular, have the capability to infer conclusions from premises which are imprecise, incomplete or not totally reliable.

Since the knowledge base of an expert system is a collection of human knowledge, and since most of human knowledge is imprecise in nature, it is usually the case that the knowledge base of an expert system consists of facts and rules which, for the most part, are neither totally certain, nor totally consistent. As a general principle, the uncertainty of information in the knowledge base of an expert system induces some uncertainty in the validity of its conclusions. Hence, to serve as a useful tool, the answer to a question must be associated, either explicitly or implicitly, with an assessment of its reliability or certainty. For this reason, a fundamental issue in the design of expert systems is how to equip them with a computational capability to analyze the propagation of uncertainty from the premises to the conclusion and associate a *certainty factor* with this conclusion .

1.2. Prolog for Expert Systems

Traditional computer programs in languages such as Pascal or ADA consist of sequences of imperative instructions, from which control may occasionally transfer to other such sequences. The languages used to encode such programs have little to offer to the expert system builder, since he would have to write all bookkeeping routines and deductive algorithms himself. This would divert his time and efforts from the more important task of modeling his expertise. Thus, a

programming language is needed that already embodies the powers of deductive reasoning. Preferably, the language should itself function deductively so that its mode of operation would resemble that of human logic.

One language that meets all these criteria is Prolog. Prolog programs consist of a collection of clauses, i.e. assertions and implications, describing the relationships that are to be computed. They may be regarded as statements of truth: the *declarative view* or they may be thought of as procedures for computing these relationships: the *procedural view*. The Prolog interpreter is an inference engine which can compute complex relationships from these simple clauses. This makes it an ideal language for modeling human reasoning and implementing expert systems.

In all expert systems, it is important that the user understands how the system is reasoning and how it reaches its conclusions. This will greatly increase the user's confidence in the expert system's decisions. Because deductive reasoning is intrinsically comprehensible, all that is needed to enable the human understanding of an expert system's execution, is a display of its path of reasoning. When a virtual relation has been built from a series of implications in the knowledge base, the system only needs to backtrack over these relations, displaying the rules at each of its nodes, in order to explain its thinking to the user. Most existing expert systems provide commands such as "HOW" and "WHY" which will initiate such machine explanations.

As a Prolog interpreter is best described in terms of Prolog itself; adding an explanation facility to a Prolog-based expert system is then a natural extension of the given reasoning strategy.

1.3. Overview of the Thesis

This thesis investigates various methodologies that have been applied to handle the modeling of uncertainty in expert systems. As Prolog is a natural choice to implement an expert system, we experimented with some of the most important approaches by programming them in Prolog.

The ideas in this thesis are developed in a top-down manner. We try to make the reader from the start familiar with the terminology and issues of inexact reasoning in expert systems. Rather than describing each methodology separately from the beginning, we considered it essential to make comparisons as early as possible in order to understand better the fundamental differences between various systems.

In the chapter that follows, we give a brief introduction to the Prolog programming language. This research involved a lot of programming as we wanted to experiment with the programming aspects of different approaches to inexact reasoning. A result is that the thesis contains a lot of Prolog code. In the interest of completeness, as the idea of deductive reasoning is central both to Prolog and to expert systems in general, we considered it to be necessary to include a brief overview of Prolog to show this duality. Prolog is a natural choice to implement expert systems.

Chapter 3 provides a general introduction to the issue of inexact reasoning in the context of expert systems. We briefly describe the most important methods. This chapter serves as an initial exploration into the world of fuzzy reasoning. This thesis essentially focuses around three approaches which in the author's opinion are the most important. The first two are included because of their historical significance and their influence on today's new expert systems. They are the *Bayesian* approach to inexact reasoning, used in the Prospector system, and the *theory of belief* of the medical diagnosis system Mycin. The third approach, *fuzzy logic* has been chosen as it holds the most promise for the future. We

demonstrate this in our thesis. After going through chapters 2 and 3, the reader should be ready for the more technical chapters of the rest of this work.

The next chapter gives an outline of implementation approaches to inexact reasoning specific for Prolog, and discusses the usefulness of the so-called *meta-level* approach.

In chapter 5, we give a detailed description of the *Bayesian* theory of conditional probabilities. This treatment is rather thorough as it is the first reasoning methodology that we implemented. After a theoretical exposure, we comment on implementation issues.

Chapter 6 provides a detailed overview of Mycin's theory of measures of belief and its Prolog implementation. In contrast with the Bayesian implementation of the previous chapter, this Prolog program is an *object-level* realization of Mycin's reasoning strategy.

The next chapter discusses important differences between the three predominant probabilistic approaches. This thesis makes a clear distinction between so-called probabilistic methods, which are all based on some notion of probability, and the non-probabilistic ones such as fuzzy logic. In this chapter 7, we bundled our critique on the various probabilistic approaches.

In chapter 8, we discuss what we feel is the most natural approach, namely fuzzy logic. We start with a technical treatment of the theory of *fuzzy sets*, the underlying analytical building blocks of fuzzy logic. We devote a section to linguistic variables, as the major goal of fuzzy logic is to capture the fuzziness of natural language concepts. An important effort was the implementation of fuzzy set operators and a fuzzy database in Prolog. This work is described in the further sections of this chapter. We also argue for the use of fuzzy logic from a psychological point of view.

The final chapter serves both as a retrospection on this thesis work and as a

prospection for future research work.

At the end of this work, we included an extensive bibliography and complete listings of the Prolog implementations of Prospector, Mycin and a fuzzy logic-based system. None of these programs is complete but each contains the complete inexact reasoning component of one system.

CHAPTER 2

THE PROLOG PROGRAMMING LANGUAGE

2.1. Introduction

In this thesis, we assume that the reader is familiar with the logic programming language Prolog. Prolog is a simple but powerful programming language for symbolic computation based on a computationally treatable subset of first-order logic, the so-called *Horn* subset, named after Alfred Horn.⁵⁵ It is a clean combination of the concepts of symbolic programming languages such as LISP and those of relational databases.⁸¹ For a good introduction to the language, we refer to Clocksin and Young²⁸ and the textbook by Clocksin and Mellish.²⁷

Prolog might be called a European programming language.^{88,10} It was born at the University of Marseille in the early seventies. After eight years of being known only to a small community of dedicated implementors and users, mainly at the University of Edinburgh, Prolog has been brought to the attention of the wider world, in particular the US,⁷⁴ by its surprising adoption as the starting point for the Japanese "Fifth Generation" computer research effort.¹⁰⁴ Prolog has been used for a variety of applications:

- natural language interaction with computer systems;^{82,29,30,32,95}
- architectural design and site planning;^{71,96,13}
- drug design (very successful commercial application in Hungary) and biomedical analysis;³⁵
- VLSI circuit analysis;^{112,9}
- various fields of artificial intelligence research;^{21,92,19}
- compiler writing (Prolog itself, APL);¹⁰³
- algebraic computation applications;^{12,14,15,20}
- database access and data description language;^{3,16,33,34,48,60}

- discrete event simulation;⁴⁹
- program development systems;⁴³
- expert systems;^{39, 77, 90, 44, 76, 52, 26}
- plan formation in robotics;^{100, 101}

The list of application areas of Prolog is growing every day. A side effect of this thesis was the creation of a database of 1200 references on Prolog and logic programming. This collection has been submitted for publication.

In this chapter, we will briefly review the Horn clause logic on which Prolog is based. We focus on the key features of the language :

1. Horn clauses;
2. Declarative versus procedural programming;
3. No explicit input/output notions;
4. Deduction, unification and backtracking.

The main part, however, will be devoted to Prolog's relationship with expert system technology.

2.2. The Horn Clause Subset of Logic

The similarity between the use of logic for representing programs and as a database language can be seen most clearly with the *Horn clause* subset of logic.^{60, 61, 62} Horn clauses provide the basis of the Prolog logic programming language.^{88, 27}

Before we deal with Horn clauses, we should first explain what clauses are. A *clause* is an expression of the form

$$B_1, B_2, \dots, B_m \leftarrow A_1, A_2, \dots, A_n$$

where B_1, \dots, B_m are alternative *conclusions* and A_1, \dots, A_n are *conditions* of the

clause. In the above clause, the arrow “ \rightarrow ” reads “*if*” to indicate logical implication. If the conclusion part of a clause contains variables, we assume that those variables are universally quantified. Variables in the conclusion part of a clause might also appear in the condition part. Variables that appear only in the condition part are existentially quantified. If x_1, \dots, x_k are variables in the conclusion part of the above clause, the interpretation is as follows:

$$\forall x_1, \dots, x_k \quad B_1 \text{ or } \dots \text{ or } B_m \text{ if} \\ A_1 \text{ and } \dots \text{ and } A_n$$

A *Horn* clause is a special case of a general clause in which there is *at most* one conclusion B . A Horn clause is thus an expression of the form

$$B \rightarrow A_1, A_2, \dots, A_n$$

In the above format, we distinguish between unit clauses ($n=0$) and non-unit clauses ($n \neq 0$).

A Prolog program, also sometimes labeled database, is a collection of Horn clauses that express information which can be used to solve problems. A *Horn clause* is either

1. A *unit clause*, also called a *fact*.

For example,

father(john,mary).

is a representation of the fact that john is the father of mary. The constants “john” and “mary” are called *atoms* in Prolog and have to start with a lower-case character. The body of a Horn clause is its condition part, which in the case of a unit clause is empty.

2. A *non-unit clause*, also called a *rule*.

For example,

mortal(X) :- man(X),old(X).

represents the idea that every old man is mortal or in other words:

X is mortal if X is a man and X is old

or, in plain English,

every old man is mortal

The symbol "X" is an example of a Prolog *variable* and has to begin with a capital. The symbol ":-" reads as "if" and the comma "," means "and"

A Prolog rule has the general format of a Horn clause:

$$B :- A_1, A_2, \dots, A_n \quad (n \neq 0)$$

where B is the *head* of the rule and A_1, \dots, A_n form the *tail* or *body*. A_1, \dots, A_n are called atomic conditions. The procedural interpretation is that in order to solve B one has to solve the conjunction A_1, \dots, A_n . Facts and rules are used to represent knowledge which can then be queried.

3. In general, a *query*, also called *question* or *program invocation*, is a collection of atomic conditions, written as

$$A_1, A_2, \dots,$$

which is interpreted as the conjunction of goals A_i of finding a substitution of terms for the variables in the query such that all of the resulting conditions are implied by the database (program) either directly by means of a fact or indirectly through a rule.

As an example, let us consider the following set of family relationships (comments are preceded by a *):

```

% facts

father(john,mary). % parents of mary
mother(judy,mary).

mother(jane,john). % parents of john
father(jack,john).

% rules

grandparent(X,Y) :- parent(X,Z),parent(Z,Y).
grandfather(X,Y) :- father(X,Z),parent(Z,Y).
grandmother(X,Y) :- mother(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y). % mother and father are
parent(X,Y) :- father(X,Y). % both parents

```

Variables that appear both in the head and the body of a rule are universally quantified where variables that only appear in the body are existentially quantified.

Given this program, we could ask the following questions:

Who is the father of mary ?

with translates into the Prolog query:

```
?- father(X,mary).
```

The Prolog system will answer this question with

```
X = john
```

This process is called *unification* of the goal `father(X,mary)` with the clause `father(john,mary)` which results recursively in the unification of the variable `x` with the atom `john`. If we now type a semi-colon (“;”) Prolog will try to find another solution by *backtracking*. In this case, no other answer is possible and the system will respond with

```
no
```

Let us now ask the question “who are the parents of mary ?”


```
?- parent(X,mary).  
X = judy ;  
X = john ;  
no
```

We should note at this point that Prolog, while answering questions, always starts its search at the top of the internal database. This is the reason why the answer “judy” appeared before “john.”

The basic computational mechanism of Prolog is top-down search through a tree of goals. Another question on the given database might be:

Who are the grandparents of mary ?

or in the Prolog notation,

```
?- grandparent(X,mary).  
X = jane ;  
X = jack ;  
no
```

Note that we only get two answers here as nothing is known about the parents of mary’s mother, judy.

One of the powerful ideas behind the Prolog programming language is that a program can be viewed in two different ways: a *declarative* and a *procedural* way.

- On one hand, a Prolog program can be viewed simply as a collection of statements, i.e. facts and rules. This is the *declarative interpretation*.
- On the other hand, the same program can also be understood as a number of procedure definitions: *the procedural view*.

As an example, let us consider one of the rules of our previous “family” relationship example:

```
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

We can read these two clauses, *declaratively*, as “X is a parent of Y if X is a mother of Y or if X is a father of Y.” In the *procedural* interpretation, they read in two different ways: “In order to find an X that is a parent of Y, find an X that is either the mother of Y or the father of Y” and “In order to find a Y such that X is a parent of Y, find a Y so that X is either the mother or the father of Y.”

We see that there are even two procedural interpretations possible. The two clauses of the example constitute together one procedure *parent* that can be used in two different ways, namely to find a parent of given child, e.g. *parent(X,mary)* or to find a child of a given parent, e.g. *parent(john,X)*. The *parent* procedure can even be used in a third way, namely to generate all instances of the (*parent,child*) relation. This effect is realized by the goal *parent(X,Y)*.

A common example of this *multi-directionality* of Prolog procedures is the *append* procedure which concatenates two lists into one list.¹⁰²

```
append([],List,List).
append([X|List1],List2,[X|List3]) :- append(List1,List2,List3).
```

We read these two Prolog clauses, in a declarative way, as “the empty list, denoted by “[]” appended to a List yields the same List” and “a list of the form X followed by a List1 appended to a List2 results in a list of the form X followed by a List3 where List1 appended to List2 gives List3.” A Prolog list is either the atom “[]” or a structure denoted as “[H|T]” where H is the first element of the list (In LISP terminology, the “car”) and T is a variable standing for the remainder or tail (“cdr”) of the list.

Now, executing the procedure call

```
?- append([a,b],[c],L).
```

will result in the answer

```
L = [a,b,c]
```

which is, of course, the concatenation of [a,b] and [c].

However, we can also use the same `append` procedure to find all lists L1 and L2 that, when appended together, result in [a,b,c].

?- `append(L1,L2,[a,b,c])`.

L1 = [] L2 = [a,b,c] ;

L1 = [a] L2 = [b,c] ;

L1 = [a,b] L2 = [c] ;

L1 = [a,b,c] L2 = [] ;

no

We conclude that Prolog avoids the notion of input and output arguments of ADA, Pascal or other purely procedural programming languages.

2.3. Prolog and Expert Systems

The knowledge needed for an expert system is usually held in the form of facts and IF-THEN rules. The expert system itself is a program, most often in LISP, that infers advice from the facts and the rules. In Prolog, a program consists of facts and rules, so the knowledge needed for an expert system can be written down directly. To get advice, one can either use Prolog's built-in inference mechanism, or one can write further rules about how to use the given knowledge. The facts in Prolog correspond to a relational database, while the rules contain expertise about how to use the data.

As Prolog treats programs and data in a similar manner, it is a perfect tool for the implementation of knowledge-based systems.

Many researchers, especially in Europe, find Prolog well-suited for implementing knowledge-based expert systems.^{26,52,99}

An expert system generally consists of three parts:

1. *A database of knowledge* (facts and rules) about a particular domain.

2. An *inference engine* for generating and controlling logical deductions.
3. An *interface* for communicating with the user.

Prolog itself provides a powerful database and one kind of inference mechanism, namely backtracking. Other types of control have been implemented.¹⁹ A user-friendly interface is easily implementable in Prolog, using one of the various logic grammar formalisms, which will not be discussed in this thesis. We will elaborate further on the architecture of expert systems in the next section.

The mostly used programming language to implement expert system is LISP. The reasons for this are, in our opinion, rather historical. As we have seen, Prolog has all the basic constructs that are needed to implement a production system. This is not the case for LISP which does not have pattern matching and search algorithms predefined. The recent efforts to implement logic programming in LISP demonstrates the need for an alternative programming methodology.^{59,87}

To summarize, we can say that programming in Prolog may be viewed as:

1. specifying *facts* about objects and their relationships
2. specifying *rules* about objects and their relationships
3. asking *questions* about objects and their relationships

CHAPTER 3

INEXACT REASONING IN EXPERT SYSTEMS

3.1. Introduction

In this chapter, we will give a general introduction to the area of expert systems and the most important methods for inexact reasoning.

An expert system has been defined as a computing system that embodies organized knowledge concerning some specific area of human expertise, sufficient to perform as a skillful and cost-effective consultant. Thus it is a high-performance special-purpose system that is designed to capture the skill of an expert consultant such as doctor of medicine, a chemist or a mechanical engineer.

Expert systems differ considerably from one another in terms of system design and capabilities, insofar as the term "expert system" itself is not yet precisely defined. However, in practice, the most important systems have many features in common:

1. A database of *facts*.
2. A database of *inference rules*, i.e. the *knowledge base*.
3. A *control strategy*

Most expert knowledge, however, is of an ill-defined and heuristic nature, frequently formulated at an subconscious level. It is most unlikely that standard text books will give sufficiently detailed, precise, and accurate information suitable for direct use. Much of the expert's skill will undoubtedly be hidden in approximate *rules of thumb*, which are seldom or never recorded. It is the task of the *knowledge engineer*, i.e. the system implementor, to extract this knowledge and to organize it. To achieve this, a period of intensive discussion with one or more experts in the subject and the analysis of a selected set of test cases, is generally needed.

Expert knowledge, i.e. *expertise*, is usually not encoded in a procedural fashion, i.e. as procedures and functions in a particular programming language, but in a declarative form, known as a *knowledge base*. This knowledge base comprises a modular set of rules of the form

situation - action.

for example

```
if the car has a battery discharge problem
and the battery has just been replaced
then consider testing the charging system
```

Each such rule is intended to correspond in a natural and direct fashion to a chunk of knowledge meaningful to the domain expert.

In addition to a knowledge base of rules, a mechanism is needed for manipulating these rules to form inferences, make diagnoses, etc. This mechanism, which is essentially a form of theorem proving, is often referred to as the *inference engine*. A design aim is to keep the inference engine standard across a wide range of systems and applications. It should be kept separate from the knowledge base, which is domain-specific.

An important complication arises from the fact that expert knowledge in many domains, e.g. medicine, is not precisely defined, but of an inherently imprecise nature such as

```
if:   the infection is primary-bacteremia and
      the site of the culture is one of sterile sites and
      the suspected portal of entry of the organism is the
      gastro-intestinal tract

then: there is suggestive evidence (0.7) that the identity of
      the organism is bacteroides
```

Many systems permit imprecise rules to be included in their knowledge base and provide means of manipulating them. This is often described as *inexact reasoning*,

plausible reasoning or reasoning with uncertainty.

The recommendations made by an expert system will be accepted by the user only if this user understands and accepts the underlying reasoning. Many expert systems provide an *explanation* facility which enables the user to consult the system interactively to ask for explanations of the system's decisions, diagnoses, etc. and the *lines of reasoning* used to obtain them.

A *user-friendly* interface is of great importance to the acceptance of an expert system. However, this does not necessarily imply the need to handle unrestricted natural language. Communication in a restricted form, in the jargon of the field, may be fully justified and even more preferred.

An important way in which expert system rules can be incorporated into a working program is in the form of a *production system*. A production system is a collection of rules of the form "*conditions - action*," where the *conditions* are statements about the contents of a global database of *facts* and the actions are procedures which may modify the contents of the database. Program execution consists solely of a continuous sequence of *cycles* terminating when some halting action is executed. At each cycle, all rules with conditions that are satisfied by the contents of the database are determined. If there is more than one such rule, a selection is made by means of some pre-determined *conflict resolution* strategy. All the actions associated with the selected rule are then executed and the database is changed accordingly.

Although expert systems are now becoming commercially available, there are still some fundamental issues in this area that need to be resolved. One of them is using a production system architecture to model the knowledge base. We mean by this the almost universal assumption that expert knowledge is best modeled using IF-THEN rules. Another, more fundamental issue which still requires a lot of research is the role of inexact reasoning. Each expert system has adopted its own method, some of them backed by a formal theory, others purely ad hoc.

3.2. Inexact Reasoning

In this thesis, we investigate the most common methods for inexact reasoning, we compare and evaluate them and describe an implementation in Prolog.

The techniques to deal with *uncertain* and/or *fuzzy knowledge* can be classified as follows:

1. *Non-monotonic reasoning* (default reasoning)^{73,86}
2. *Probabilistic reasoning*^{41,42,94,73}
3. *Fuzzy logic* (Possibilistic Reasoning)¹⁰⁷

Non-monotonic logic has not been used yet in any existing expert system; it is more oriented towards natural language processing systems. We do not treat it here but rather concentrate on the *probabilistic* approaches and on the *fuzzy logic* method. We will compare the various methodologies that were implemented in real systems, or that were proposed as a methodology.^{78,109}

The class of *probabilistic strategies* includes :

- *Mycin-like inexact reasoning* (measures of belief and disbelief)
- *Prospector-like inexact reasoning* (subjective Bayesian updating)
- *Dempster-Schafer theory*

Each of these will be discussed in the subsequent chapters. First, we will formulate the problem of inexact reasoning in general terms.

In a binary problem, each symbol is assumed to be either completely true or completely false, and only two relationships in the decomposition of a problem are possible: AND and OR. However, in real-world problems, two kinds of *uncertainties* exist:

1. The uncertainty associated with the observed evidence E .
2. The uncertainty of the expressed knowledge itself, i.e. the uncertainty of the inference rule $E - H$ (read : if E (evidence) then H (hypothesis)).

Thus, an AND/OR tree is insufficient for most of the real-world problem descriptions. Expert systems commonly distinguish between three kinds of rules which all have uncertainty associated with them:

1. AND-rules

if E_i and E_j then H with C_i

2. OR-rules

if E_i or E_j then H with C_i

3. Multiple Evidence rules

if E_i then H with C_i

if E_j then H with C_j

In the above formulae, we used

E_i, E_j as pieces of evidence

H_i for hypothesis

C_i as certainty measures

We note that if E_i and E_j in an OR-rule are independent of each other, then the rule becomes a multiple evidence rule.

Expert systems commonly employ some means of drawing inferences from domain and problem knowledge, where both the knowledge and its implications are less than certain. Methods used to express this uncertainty include, among others, *subjective Bayesian reasoning* (Prospector), *measures of belief and disbelief* (Mycin) and the *Dempster-Schafer theory of evidence*.

Thus, when expert systems are applied to real-world situations, and not to idealized theoretical worlds, Boolean logic is no longer sufficient. The first techniques that were used to deal with uncertainty were based on *Bayes' rule of conditional probabilities*, which is part of the traditional probability theory.

Probability theory is the approach most widely used for the representation and manipulation of uncertain information. If we assume that we have, in general, an *inference rule*

$$E \rightarrow H$$

then, using probability theory, we may attribute a probability P to both E and H and define a *conditional probability* as

$$P(H | E) = \frac{P(H \& E)}{P(E)} \quad (\text{Bayes rule})$$

$P(H \& E)$ is called a *joint probability*. $P(E)$ is a *prior probability*. The basic axioms of probability, together with the above rule for conditional probability, are the basis for the *Bayesian probability theory*, named after the 18th century British mathematician Thomas Bayes.

We note that the specification of a value of the conditional probability $P(H | E)$ is the way to express the validity of the implication

if E then H

The success of Bayes' theorem in early expert systems was due largely to the vast amounts of data available for their domain of application, mostly specific medical fields or geology, as in *Prospector*. However, a Bayesian approach in domains where the appropriate data are not available necessitates the use of numerous approximations and assumptions. The *subjective probabilities*, provided by the *domain expert*, replace the statistically estimated probabilities. Obviously, with subjective estimates, it is hard to guarantee consistency. We will explain this further in the chapter on *Prospector*.

The inadequacies of Bayesian probability in the analysis of some real-world problems has led to various other approaches. These include the theory of *fuzzy sets*¹⁰⁷, the theory of *confirmation*^{22,94}, the theory of *upper and lower probabilities*

37,38 and the theory of *evidence*⁹¹ Each of these provides an alternative basis for mechanizing inferential reasoning. We will treat each of them separately in this chapter.

3.3. Mycin

*Mycin*⁹⁴ is a program that attempts to recommend appropriate therapies for patients with bacterial infections. *Medical diagnosis* is an example of the sort of real-world task that defies absolute analysis because of its complexity and lack of complete knowledge. In such domains, inexact reasoning is necessary.

According to the authors of the Mycin system, Bayes' theorem could not be used for two major reasons:

1. It is often difficult to collect all the prior probabilities in the field of medical diagnosis.
2. Bayes' theorem is too static and too expensive when new knowledge, i.e. facts and rules, are acquired. In particular, most of the joint probabilities would have to be recomputed as the sum of the probabilities of the possible outcomes has to be equal to one.

Let us formulate, in general, a medical diagnosis problem using Bayes' theorem. We adapt the following notations:

D_i = *i*th diagnosis (disease)

E = sum of all the relevant data, total evidence

Then

$P(D_i | E)$ = the conditional probability that the patient has
disease *i* in the light of the evidence E

To be able to compute this conditional probability, i.e. solve the inference problem, we try to use Bayes' formula

$$P(D_i | E) = \frac{P(D_i \& E)}{P(E)}$$

We note that the domain expert has to provide a prior probability $P(E)$ for each proposition E , which is difficult in practice and that the joint probabilities $P(D_i \& E)$ have to be known for each (E, D_i) combination. In other words, Bayes' approach to medical diagnosis would require a huge amount of data. This idea convinced the authors of Mycin to use another approach in their system.

In Mycin, the domain expert provides a *certainty factor* (CF) for each inference rule $E \rightarrow H$. This factor may vary on a scale of -1.0 to 1.0 where

-1.0 = complete confidence that H is false when evidence E is true

1.0 = complete confidence that H is true when evidence E is true

0 = no information (H is independent of E)

The certainty factor captures the expertise of the expert system builder. Propositions in Mycin have two parameters associated with them, a *measure of belief* and a *measure of disbelief*. These measures are based on *Carnap's theory of confirmation*²² and are defined as follows:

$MB(H, E)$ = measure of belief in H for given E , i.e., the degree to which
the hypothesis H is supported by an observed evidence E

$MD(H, E)$ = measure of disbelief in H for given E , i.e., the degree to which
the hypothesis H is negated by an observed evidence E

These two measures can be defined formally in terms of the Bayes conditional probability $P(H | E)$ and the prior probability $P(H)$. The *certainty factor* is defined in terms of measures of belief and disbelief as follows:

$$CF(H, E) = MB(H, E) - MD(H, E)$$

$CF(H,E)$, $MB(H,E)$ and $MD(H,E)$ are judgemental measures that reflect a particular model of belief. They are *not* probabilities as the general rule

$$CF(H,E) = 1 - CF(not\ H,E)$$

does *not* hold. The CF of a hypothesis is used to compare the *evidential strength* of competing hypotheses.

According to the authors of the Mycin system, their proposed model for inexact reasoning is a plausible representation of the numbers an expert gives when asked to quantify the *strength* of his judgemental rules. We quote from Shortliffe⁹⁴

“The expert gives a positive number ($CF > 0$) if the hypothesis is confirmed by the observed evidence, suggests a negative number ($CF < 0$) when the evidence suggests the negation of the hypothesis and says there is no evidence at all ($CF = 0$) if the observation is independent of the hypothesis under consideration.”

The $CF(H,E)$ combines knowledge of both the prior probability $P(E)$ and the Bayes conditional probability $P(H | E)$. Since the medical expert often has trouble stating $P(E)$ and $P(H | E)$ in quantitative terms, a $CF(H,E)$ that weights both numbers is a more natural *intuitive* concept. Certainty factors in Mycin play the role of the conditional probabilities in Bayes' theorem. Mycin's certainty factors reflect the idea that medical diagnosis is based on *judgemental data* rather than on statistical data as in the Bayes approach, used in the Prospector system.

Thus, Mycin,⁹⁴ which is one of the most successful and influential expert systems, is based on a theory different from Bayes', namely the *theory of confirmation*. The approach here was, in fact, to develop a system that was both based on theoretical results and of practical use. Whenever this was not possible, intuitively motivated techniques were added. Mycin's knowledge base is a set of rules, each consisting of one or more stimulus propositions (*evidence*), a response

proposition (*hypothesis*) and a *certainty factor* that quantifies the degree to which belief in the stimulus propositions confirms the response proposition. All this information is provided by the domain expert, who sets up the expert system.

A certainty factor combines two measures from the theory of confirmation, *the measure of belief* and *the measure of disbelief*. The justification for their approach is not to attempt to improve on Bayes' theorem but rather to develop a mechanism whereby judgemental knowledge can be efficiently represented and utilized for the modeling of medical decision making. This is especially valid in contexts where

1. statistical data are lacking
2. probabilities of the negation are not known or not guaranteed
3. conditional independence can be assumed in most cases

Mycin's success in diagnosing bacterial infections, despite a lack of statistical data, suggested that similar techniques might be advantageously employed in other domains with a shortage of statistical data. This resulted in the EMycin system that was ported to other domains.^{17,2} EMycin stands for Essential Mycin, i.e. Mycin stripped of its domain knowledge.⁷⁵

Finally, as we will see later, propagation of uncertainty in Mycin uses formulas, similar to those in Prospector, and also various approximations to guarantee consistency within the given model. We should also note that Mycin only uses inference rules with a *CF* above a predefined threshold.

3.4. Prospector

Mycin's use of production rules to represent judgemental knowledge and its inclusion of formally based mechanisms for handling uncertainty influenced the design of Prospector,^{41,84} a geological consultant system intended to help geologists in evaluating the mineral potential of exploration sites. However,

Prospector's inference mechanism is not based on the theory of confirmation, but on a *subjective Bayesian* technique that retains, insofar as possible, the well-understood methods of Bayesian probability theory, introducing only those modifications needed to compensate for the subjectivity of the probabilities. Subjective probabilities are interpreted as measuring degrees of belief rather than long-run relative frequencies of occurrence. Prospector uses *approximations* to overcome many of the problems of dealing with subjective probabilities. Some of these approximations include:

1. The use of a piecewise linear interpolation function to correct for inconsistent probabilities, i.e. probabilities that do not conform with Bayes' rule. The definition of this function is, as the authors admit themselves, pretty ad hoc.
2. The assumption that evidence combines either independently or as a logical function, i.e. conjunction, disjunction or negation.
3. An interpolation formula to account for the combination of uncertain independent evidence.
4. The use of simple formulas from fuzzy set theory to combine dependent evidence.

These approximations led to a computationally simple method for updating probabilities that has proven to be very successful in practice.^{41,50}

Prospector's formulation offers several advantages over that of Mycin. Since Bayesian probability theory is the most widely known, Prospector's formulation often creates fewer conceptual barriers. Unlike Mycin, Prospector can utilize a rule regardless of the level of support that exists for its stimulus propositions. Mycin can only use a rule once its stimulus propositions have reached a level of support above a preset, empirically selected threshold. Thus, Prospector makes more complete use of the available information than Mycin does. We will go into more detail on this subject when we discuss the inference formulas Prospector

uses in chapter 5 .

Although both Mycin and Prospector represent giant steps towards a well-founded theory of mechanized inexact reasoning, they share some common problems. A number of these problems center around their *lack of internal consistency*. Both Mycin and Prospector partially correct for these problems by constraining their inferencing process. Inference is restricted to a single, predetermined direction along each arc within the inference graph. No loops are permitted. However, it seems to us that such loops are part of human reasoning. Another major criticism on Prospector has been its single-valued probabilities.³ Mycin uses an alternative approach in which there are two separate values for the validity of each proposition. However, this two-valued approach is also subject to the criticism about precision of the actual estimates, since both of the belief measures Mycin is using are also point values. We refer to chapter 7 for a comprehensive comparison between the two systems.

3.5. The Dempster-Shafer Theory

A mathematical theory of evidence and an accompanying theory of probabilistic reasoning by Shafer⁹¹ provides an alternate foundation for the construction of mechanized systems for inexact reasoning. Shafer's work is an extension of Dempster's³⁷ on *partial belief*. Shafer's theory departs from the more traditional Bayesian theory¹¹ avoiding several of its documented difficulties, especially its inability to represent ignorance and its insistence that new evidence be expressible as a certainty. Shafer's theory is fundamentally different from the theories underlying Mycin and Prospector. Unlike the theory of confirmation and the Bayesian theorem, Shafer's theory does *not* rely on prior subjective probabilities. It takes a conservative view: inferences are made by eliminating the impossible, not by assuming the probable.

Sets of confidences are associated with propositions. It is presumed that the

true confidence of each proposition is an element of its associated set. Inferencing consists of reducing these sets by eliminating those elements that are inconsistent with the sets assigned to related propositions. Shafer's theory is the basis for dependency-graph models of evidential support.⁶⁸ *Dependency-graph models of evidential support* are formal systems capable of pooling and extending evidential information, while maintaining internal consistency. In this formalism, the likelihood of a proposition is represented as a subinterval of the unit interval. The lower bound represents the *degree of support* and the upper bound stand for the extent to which the proposition remains *plausible*. Evidential information is in the form of *mass distributions* which are collected from different sources of knowledge and combined through Dempster's rule of combination.³⁷ Using this theory, prior probabilities are no longer needed. The Dempster-Shafer theory has, to the author's knowledge, only been employed in the VISIONS system.¹⁰⁵

Ishizuka⁵⁷ discusses an extension of the basic Dempster-Schafer theory to include *fuzzy expressions*, i.e. fuzzy predicates and fuzzy certainty. Basically, he describes a system based on two theories:

1. Dempster-Schafer theory
 - a. Probabilities are seen as mass distributions.
 - b. The calculation of the probability of a hypothesis under multiple evidence uses Dempster's rule of combination.
2. Fuzzy logic
 - a. The natural language constructs in the propositions also have a measure of fuzziness.
 - b. The attributed degrees of uncertainty may be fuzzy themselves.

Although the Dempster-Schafer theory overcomes many of the inadequacies of the single-valued probabilistic schemes, it still suffers from some problems:

1. It is difficult for people to think in terms of probabilities.

2. The Dempster-Schafer detects conflicting hypotheses but cannot resolve them.

As fuzzy logic extends the capabilities of the Dempster-Schafer theory, we decided to concentrate our efforts on the former.

3.6. Fuzzy Logic

Another approach to inexact reasoning that diverges from classical logic is *fuzzy logic*, as discussed by Zadeh and others.^{107,110,4,70}

In existing expert systems, as we have mentioned previously, the computation of certainty factors or similar measures is carried out through a combination of methods which are based on, or at least, similar to, two-valued logic and probability theory. However, these methods have serious shortcomings that are commented throughout this thesis. In particular, one open question is the universally made assumption that if each premise is associated with a numerical certainty factor then the certainty factor of the conclusion is a number which may be expressed as a function of the certainty factors of its premises. Zadeh claims that this assumption is in general invalid because it does not capture all the uncertainty of the rule as we will show below.¹⁰⁹

He argues that the employment of fuzzy logic as a framework for the management of uncertainty in expert systems makes it possible to consider a number of issues which cannot be dealt with effectively or correctly by conventional methods. According to him, the more important of these are the following:

1. The fuzziness of antecedents and/or consequents in rules of the form

(a) if X is A then Y is B

(b) if X is A then Y is B with $CF = \alpha$

where the antecedent, X is A , and the consequent, Y is B , are fuzzy propositions, and α is a numerical value of the certainty factor CF . For example,

if *X is small then Y is large* with $CF = \alpha$

in which the antecedent *X is small* and the consequent *Y is large* are fuzzy propositions.

2. Partial match between the antecedent of a rule and a fact supplied by the user.

Since typically, the number of rules in an expert system is relatively small, i.e. of the order of a few hundred, there are likely to be many cases in which a fact does not exactly match the antecedent.

3. The presence of fuzzy quantifiers in the antecedent and/or consequent of a rule.

In many cases, the antecedent and/or consequent of a rule contain implicit or explicit fuzzy quantifiers such as *most, many, few, etc.*

Zadeh¹⁰⁹ claims that most of the facts and the rules in a real-world expert system contain fuzzy predicates and are thus fuzzy propositions. In the existing expert systems, he continues, the fuzziness of the knowledge base is ignored because neither predicate logic nor probability-based methods provide a systematic basis for dealing with it. For example, one of the traditional laws of probability theory

$$P(H | E) = 1 - P(\text{not } H | E)$$

does *not* hold when *E* is a fuzzy proposition. Zadeh defines a knowledge base as a collection of propositions of one of the following *canonical forms*:

1. An unconditional, unqualified proposition, i.e. fuzzy fact

e.g. *John has a young daughter*

young is a fuzzy predicate

2. An unconditional, qualified proposition, i.e. fuzzy fact

e.g. *it is very likely that John has a young daughter*

young is a fuzzy predicate

very likely is a fuzzy probability

3. A conditional, unqualified proposition, i.e. fuzzy rule

e.g. *if X is young then X is good-looking*

young and *good-looking* are fuzzy predicates

4. A conditional, qualified proposition, i.e. fuzzy rule

e.g. *if a car is old then it is probably not very reliable*

old and *reliable* are fuzzy predicates

probably is a fuzzy probability

not very is a fuzzy quantifier

In order to be able to convert any natural language proposition into its canonical form, Zadeh devised what he calls *test-score semantics*. For example, the *fuzzy fact*

John has dark hair

has a canonical form *X is F* where $X = \text{color}(\text{hair}(\text{John}))$ and $F = \text{dark}$, a fuzzy subset of the set of colors of human hair.

Canonical forms are also able to deal with conjunctions, disjunctions and negation, based on fuzzy set theory. We will treat fuzzy set theory in depth in chapter 8. Zadeh shows¹⁰⁸ that any natural language proposition may be expressed in a canonical form by the use of test-score semantics. We will not elaborate on this.

Each fuzzy concept has a *possibility distribution* associated with it. This accounts both for the propositions and for the certainty factors. Zadeh proves that the problem of drawing inferences is, by the use of fuzzy logic, reduced to

that of solving a *nonlinear program*

$$D \leq f(R) \text{ with local and global constraints}$$

where f is a nonlinear function and D and R are matrices of variables. We will elaborate on this in chapter 8.

We conclude temporarily that fuzzy logic has two principal components:

1. A translation system for representing the meaning of propositions and other semantic entities in the form of possibility distributions.
2. An inference system to propagate these possibility distributions.

The theory of fuzzy logic has been applied in a variety of domains, but mostly in the fields of pattern recognition and decision-making. Fuzzy system theory has had little impact on the literature of Artificial Intelligence. Kling⁵³ and LeFaivre^{66,67} give extensions to the AI programming language, PLANNER, to allow the use of fuzzy logic. Lee⁶⁵ made a study of resolution theorem proving for a fuzzified form of predicate calculus.

We see several reasons for the fact that fuzzy logic has almost never been used in AI:

1. Fuzzy logic has a very formal and theoretical basis unlike most AI techniques which often lack theoretical foundations.
2. Fuzzy logic is difficult to implement.
3. There is a misconception that all research in fuzzy logic is conducted by one person, Zadeh.

We will go into more detail on the theory of fuzzy logic in chapter eight.

CHAPTER 4

INEXACT REASONING IN PROLOG

4.1. Introduction

A *rule-based expert system* can be regarded as a *production system*. A production system consists of three parts:

1. A *global database* of facts (pieces of evidence and deductions)
2. A set of *production rules* (inference rules, deduction rules)
3. A *control system* (search mechanism)

The set of production rules operates on the global database of facts. A production rule has the format:

$$H \text{ if } E$$

where H is called *hypothesis*, conclusion, postcondition or consequent and E is the *evidence*, premise, precondition or antecedent. The precondition E of a production rule R is either satisfied or not satisfied by the global database. By "satisfied" we mean provable. If it is satisfied, the rule can be applied, or in other words, the *inference* (deduction) can be made. Applying a rule, i.e. matching its premise pattern, changes the state of the global database. The control mechanism chooses which rule should be applied and halts the computation when a termination condition on the global database is reached.

Most expert systems are implemented as so-called *backward-chaining* production systems, i.e. reasoning backwards from the top goal to the pieces of evidence, supplied by the user in a consultation session. This approach is also called *goal-driven*, i.e. standard Prolog. Some expert systems, however, are *forward-reasoning* systems, especially those that depend on real-time data. They are also called *data-driven* expert systems.

4.2. Probabilistic Reasoning

Probability is a *calculus* for expressing *uncertainty*. This uncertainty may be due to incomplete knowledge and/or to the variability of the world the expert system is defined upon. Practically each successful expert system, that used probabilistic calculus to deal with uncertainty, took a different approach. The difference could be characterized by the meaning the different systems attribute to the probability function P . Propector adopted the *personalistic* interpretation of probability: the probability of a proposition is the confidence of an individual in the *truth* of the proposition. Mycin on the contrary uses the *confirmation* definition of probability: P is regarded as a unique relation between propositions E and H where $P(E \rightarrow H)$ denotes the *support* of E for H .

The problem we are dealing with consists of drawing inferences and conclusions from *uncertain* or *incomplete evidence*.

An *inference rule* has the general format

$$E_1 \text{ and } E_2 \dots \text{ and } E_n \rightarrow H$$

where E_i is a piece of evidence and H stands for the hypothesis. This inference rule can be abbreviated as

$$E \rightarrow H$$

where the *probability* P of E is defined, in most expert systems, as

$$P(E) = \min \{ P(E_i) \}$$

following Zadeh.¹⁰⁶ The probability of a disjunction is defined in a similar manner, i.e. as a maximum. It is still unclear to us why most existing expert systems adopted formulas from fuzzy set theory to calculate the probabilities of conjunctions and disjunctions of propositions.

Several pieces of evidence may lead to the same hypothesis, so, in general, we are dealing with an *inference net*.

There are two kinds of uncertainty:

1. The *strength* of the rule $E \rightarrow H$.
2. The *uncertainty* of a piece of evidence E or of a proposition X in general.

The distinction between these two kinds of uncertainty is important because typically only the values associated with propositions can be altered as a result of the inference.

The problem we are dealing with is the *propagation* of uncertainty through the inference network. More specifically, the inference problem consists of computing the posterior conditional probability $P(H | E)$ after the acquisition of information on evidence E .

4.3. Fundamentals of Probabilistic Reasoning in Prolog

Prolog provides a natural way for implementing a production system.

1. Facts are expressed as *Prolog unit clauses*.
2. Production rules can be directly expressed as, or transformed into nonunit *Horn clauses*.
3. Prolog provides a built-in search mechanism, namely *backtracking* (depth-first). However, other search strategies, such as breath-first, may be implemented without too much effort.

One important component of a real-world rule-based expert system that is not readily available in Prolog is some mechanism to deal with *uncertainty*. This uncertainty appears both in facts and rules in every day applications. What is needed is a way of associating probabilities with facts and rules and more importantly a general strategy for computing the uncertainty of a conclusion, given the uncertainties of all the premises. In one of the first papers on this subject, Clark and McCabe²⁶ suggest to add one extra argument to each predicate whose value is the uncertainty of the solution returned in this predicate and to augment the

body of each non-unit clause with an additional goal, whose purpose is to compute the certainty of the clause, given the certainties of the solutions to the goals in the condition (body) of the clause.

So, instead of, for example

$$h(X,Y) :- e1(X,Y), e2(X,Y)$$

we would write

$$h(X,Y,P) :- e1(X,Y,P1), e2(X,Y,P2), compute(P1,P2,P)$$

where `compute` would be the definition of the reasoning strategy followed. `P1` and `P2` would be the uncertainty of `e1` and `e2` respectively. Thus, inexact reasoning is embedded in the knowledge base itself.

However, Ehud Shapiro⁹³ suggests dealing with uncertainty in Prolog programs at the *meta-level*. This consists of augmenting the Prolog interpreter to compute probabilities while doing inferences. This technique, which is both simpler and more powerful than what Clark and McCabe suggested, will be followed in this chapter. Shapiro called his approach *logic programs with uncertainties*. The advantages of his method are:

1. Precise semantics can be given to logic programs with uncertainties.⁹³
2. Standard logic programs are a special case of logic programs with uncertainties. If all the certainty factors are equal to one, then the given semantics and interpreter for logic programs with uncertainties reduce to the standard semantics and interpreter for logic programs.
3. It is easier to define debugging algorithms on top of logic programs with uncertainties, when defined in this way.

4.4. Interpreters for Logic Programs with Uncertainties

In this section we will give a brief overview of Shapiro's approach.⁹³ A *certainty factor* is a real number, greater than zero and less than or equal to one. A *certainty function* f is a function from multisets of certainty factors to certainty factors. A *logic program with uncertainties* is a finite set of pairs $\langle A :- B, f \rangle$, where $A :- B$ is a definite clause and f is a certainty function. A multiset is essentially a Prolog list.

The certainty function is used to compute the certainty of the conclusion of a clause, given the multiset of certainties of solutions to goals in the condition of the clause. A certainty function f has to fulfill the following constraints for every multiset S :

1. $f(S \cup \{1\}) = f(S)$ (adding a true proposition)
2. $S \subseteq S1$ implies $f(S) \leq f(S1)$ (f is monotone increasing)

Now, for every pair $\langle A :- B, f \rangle$ in a logic program with uncertainties P , the number $f(\{\})$ is called the *certainty factor* of $A :- B$ in P .

An interpreter for logic programs with uncertainties may be written as a logic program. An interpreter for Prolog programs *without* uncertainties is traditionally implemented as:

```
solve(true).  
solve((A,B)) :- solve(A),solve(B).  
solve(A) :- clause(A,B),solve(B).
```

An interpreter for logic programs with uncertainties is an extension to this interpreter. It assumes that the program P is represented as clauses $\text{clause}(A,B,F)$ for any clause $A :- B$ with certainty function F in P . Multisets may be represented using Prolog lists where $[\]$ is the empty list or set and $[X|Y]$ is the list with head X and tail Y , which represents the multiset $\{X\} \cup Y$. Now, an interpreter solve for a logic program P with uncertainties could be described as:

```

solve(true,[]).
solve((A,B),[X|Y]) :- solve(A,X),solve(B,Y).
solve(A,F(S)) :- clause(A,B,F),solve(B,S).

```

The semantics of `solve(A,C)` is: "A is provable from P with certainty C." The interpreter, described above, returns in C an unevaluated expression with occurrences of certainty function symbols in it. To compute the actual certainty factor, one has to evaluate this expression. Most Prolog implementations do not allow expressions of the form `F(S)` where F is a variable symbol. This would add second-order logic constructs to Prolog. One way to get around this is to define `apply` as follows:

```

% apply(foo,[X,Y]) == foo(X,Y)
% apply(foo(X),[Y]) == foo(X,Y)

apply(Pred,Args) :- atom(Pred),
                    Goal =.. [Pred|Args],!,
                    call(Goal).

apply(Pred,Args) :- Pred =.. OldList,
                    append(OldList,Args,NewList),
                    Goal =.. NewList,!,
                    call(Goal).

```

With this definition, `F(S) == V` becomes equivalent to `apply(F,[S,V])` and we may rewrite the basic interpreter as

```

solve(true,[]).
solve((A,B),[X|Y]) :- solve(A,X),solve(B,Y).
solve(A,V) :- clause(A,B,F),solve(B,S),apply(F,[S,V]).

```

where

```

clause(A,B,F) :- clause(A,B),current_function(F).

```

The `current-function` looks up what the current certainty function is. This feature is useful for comparing the effect of different strategies. This predicate is the only side effect when we want to use the same basic interpreter for another certainty function. Using a new strategy for inexact reasoning means defining a

new certainty function.

The basic interpreter, described above, may be extended, for example, with a *certainty threshold* by putting additional constraints such as `gt(S,C)` in the body of

```
solve(A,V) :- clause(A,B,F),
              solve(B,S),
              threshold(S),
              gt(S,C),
              apply(F,[S,V]).
```

where `gt(S,C)` means that `S` is greater than some threshold `C`, which is retrieved by the call `threshold(S)`. This will prune low-certainty execution paths. Certainty thresholds are for example used in Mycin.

When we compare Shapiro's proposal with the previously mentioned `compute` predicate suggested by Clark and McCabe, we note the following:

1. Both approaches also support sets of probabilities in addition to point values.
2. Applying another inference mechanism will involve recoding of `compute` (for Clark and McCabe) or `F` (or Shapiro).

However, in Shapiro's approach, the certainty function is completely separate from the knowledge base. In the first approach, the computations depend on the inference rule itself, i.e. conjunction in the premise part. Introducing a disjunction rule cannot be easily modeled. The first approach handles conjuncts and possibly disjuncts at the object level where Shapiro deals with them at a meta-level.

It turns out that, if one would use the first approach to implement a subjective Bayesian approach, that the Prolog definition of "or" would not correspond to the logical "or".

We conclude here that indeed Shapiro's proposal provides a clear and clean strategy for introducing uncertainty in Prolog programs. However, we will question its practical use in the context of Prospector.

The major novelty, introduced in this chapter, is that we demonstrated how uncertainty can be dealt with at the meta-level. In our opinion, this is one of the most attractive features of Prolog. We have seen that one can describe a Prolog interpreter in three lines of code. Adding additional power to this interpreter, such as inexact reasoning, is equivalent to extending the interpreter with additional control features. This has the following advantages:

1. We use Prolog to implement Prolog with uncertainties.
2. The augmented interpreter has a clear and concise declarative notation.

CHAPTER 5

SUBJECTIVE BAYES STRATEGY (PROSPECTOR)

5.1. Introduction

In this section, we will give an overview of the inexact reasoning strategy that is used in the Prospector system. This strategy was implemented in Prolog and the code is included in appendix 1. Some of the material in this section is based on Reboh's dissertation.⁸⁴

Prospector is an expert system that was developed to help geologists in exploring for hard-rock mineral deposits.^{84, 85}

In *Prospector*, a *probability* is associated with every statement in the knowledge base, corresponding to a node in an inference network, which measures the degree to which the statement is believed to be true. The basis for the reasoning procedure *Prospector* uses to propagate a probability from evidence E to hypothesis H is an elementary theorem of probability theory called *Bayes' rule*. We assume for the moment that we are dealing with a general inference rule

$$H - E$$

The *Bayesian* method assumes that before any information has been entered into the expert system during a consultation session, each statement S has some *prior probability* $P(S)$. We note here that this may create some conceptual barriers.

A prior probability of a proposition is the probability of that proposition in absence of any specific evidence in favor or disfavor of that proposition.

As evidence is acquired during the consultation process, a *posterior probability* is computed. If E' denotes all the evidence about E accumulated up to some point in the consultation, then the posterior probability $P(H | E')$ denotes the current probability of H given the evidence E' . The prior probabilities are generally supplied by the domain expert at the time the model is constructed.

The formulas we used are described in Duda et al.⁴¹ and Reboh's work.⁸⁴ The basic formula is the *odds likelihood form* of Bayes' rule

$$O(H | E) = LS \times O(H) \quad (1)$$

where

$O(H)$ = *prior odds* on hypothesis H

$O(H | E)$ = *posterior odds* on hypothesis H

(given that evidence E is present)

LS = *level of sufficiency*

The odds O and probability P for any situation are related by

$$O = \frac{P}{1 - P}$$

so that odds and probabilities are equivalent. The level or measure of sufficiency LS is defined as

$$LS = \frac{P(E | H)}{P(E | \text{not } H)}$$

Thus, Bayes' rule (1) specifies that the observation of E changes the odds on H by the factor LS . The odds likelihood formula is equivalent to the conditional probability formula of Bayes' theorem. Basically, it avoids the joint probabilities.

Analogous formulas and remarks hold for the case in which the evidence E is observed to be definitely absent (*not E*), in which case Bayes' rule has the form:

$$O(H | \text{not } E) = LN \times O(H)$$

where LN is called the *level of necessity* and is defined by

$$LN = \frac{P(\text{not } E | H)}{P(\text{not } E | \text{not } H)}$$

Numerical values for the likelihood ratios, LS and LN , are also supplied by the

domain expert.

5.2. Uncertainty of Evidence

Bayes' rule can only be used when there is absolute certainty about the truthfulness or falseness of a piece of evidence. In actual practice, the user is often unable to observe either the definite presence or absence of the evidence. Typically, the user is prepared only to indicate a *degree of confidence* that the evidence sought is actually present. In this case, Prospector uses a formula that effectively interpolates between the two extreme cases of perfect certainty.

Let E' denote the observations that cause the user to suspect the presence of evidence E . The posterior probability $P(H | E')$ is thus somewhere between $P(H | \text{not } E)$ and $P(H | E)$. In Duda et al.,⁴¹ it is shown that under certain assumptions, $P(H | E')$ is a linear function of $P(E | E')$, with $P(H | E') = P(H | \text{not } E)$ when $P(E | E') = 0$ and $P(H | E') = P(H | E)$ when $P(E | E') = 1$.

Because the probability values $P(H | \text{not } E)$, $P(H | E)$, $P(H)$ and $P(E)$ are all obtained from the domain expert's subjective estimates, they might be inconsistent with the theoretically expected values. In particular, we must make certain that when nothing is known about E , i.e., when $P(E | E')$ is equal to $P(E)$, the interpolation function should leave H at its prior value, yielding the theoretically expected value $P(H | E') = P(H)$.

$P(H | E')$ is therefore chosen to be a piecewise linear function of $P(E | E')$ so that the desired values for $P(H | E')$ are obtained at the three fixed points $P(E | E') = 0$, $P(E)$ and 1. The resulting function can analytically be expressed as follows:

$$P(H | E) = \begin{cases} P(H | \text{not } E) + \frac{P(H) - P(H | \text{not } E)}{P(E)} \times P(E | E') & \text{for } 0 \leq P(E | E') < P(E) \\ P(H) + \frac{P(H | E) - P(H)}{1 - P(E)} \times P(E | E') - P(E) & \text{for } P(E) \leq P(E | E') \leq 1 \end{cases} \quad (2)$$

So, in a sense, the basic formula Prospector is using is not justified by the traditional probability theory but has proven to be an effective practical method for treating uncertain evidence.⁸⁴

5.3. Multiple Evidence

In real-world applications, a single inference rule of the form

$$H - E$$

is inadequate. Two generalizations are needed.

First of all, there might be *multiple evidence* leading to the same hypothesis. In general,

$$H - E_1$$

$$H - E_2$$

$$H - E_3$$

...

$$H - E_n$$

Prospector assumes that all the E_i are *independent*. In a simple backtracking production system, like Prolog, in order to prove H , it is sufficient to prove one of the E_i . However, in a system with inexact reasoning the certainty of a hypothesis H is updated by the probabilities of each of its pieces of evidence. The general formula Prospector uses is

$$O(H | E') = \prod_{i=1}^n L_i O(H) \quad (3)$$

where

$$L_i = \frac{O(H | E'_i)}{O(H)}$$

and is called the *effective likelihood ratio* for E_i . So, the probability of a hypothesis H with multiple pieces of evidence E_i will be updated by all of them.

A second generalization is the fact that the antecedent of an inference rule may contain a logical *combination of evidence*, for example

$$H - E_1 \text{ or } (E_2 \text{ and } E_3)$$

Prospector uses simple formulas from *fuzzy set* theory to compute the probabilities of logical combinations of evidence.

$$\begin{aligned} &\text{if } E = E_1 \text{ and } E_2 \text{ and } \dots \text{ and } E_n \\ &\text{then } P(E | E') = \min_i P(E_i | E') \quad (4) \end{aligned}$$

$$\begin{aligned} &\text{if } E = E_1 \text{ or } E_2 \text{ or } \dots \text{ or } E_n \\ &\text{then } P(E | E') = \max_i P(E_i | E') \quad (5) \end{aligned}$$

$$\begin{aligned} &\text{if } E = \text{not } E_i \\ &\text{then } P(E | E') = 1 - P(E_i | E') \quad (6) \end{aligned}$$

Almost all existing expert systems that have a fuzzy inference mechanism use equations (4),(5) and (6) to compute the probability of the AND/OR combination of evidence. These equations are generalizations of the classical truth manipulation formulas of two-valued logic. It is still unclear to us why these formulas are

preferred over others. Clearly, they are easy to compute and are in most cases a reasonable approximation.

Equation (3) is a generalization of Bayes' rule (1). Whenever the probability of an antecedent for H changes, the corresponding effective likelihood ratio changes, and the probability of H is updated by reevaluating equation (3). If H is itself the antecedent of other inference rules, or part of the antecedent of other rules, the computations are repeated to update the probabilities of these higher-level hypotheses. Thus, by repeated use of the formulas (3,4,5,6), the effects of acquiring new evidence are propagated throughout the inference network.

5.4. Conclusion

Each statement is either the logical combination of other statements in which case we use the above formulas (4,5,6) to compute its probability, or it might be the consequent of one or more inference rules in which case we use the multiple evidence rule (3).

Initially, *the domain expert* who builds the expert system has to provide

1. for each node : the prior odds (or prior probability)
2. for each arc : the effective likelihood ratio

In the consultation phase, the user of the expert system provides his belief in the evidence, by specifying $P(E_i | E_i')$, after which a new effective likelihood ratio will be calculated for each rule $H - E_i$, which will produce updated posterior odds for the final hypothesis. If the user doesn't know anything about a piece of evidence, the prior odds can be used.

A major problem in Prospector is that the domain expert has to provide all the prior probabilities.

Bayesian statistics presents an attractive approach. It provides a practical solution provided all information is available, in particular the prior probabilities

for each proposition. This is not always possible for real-world application. Moreover, people are usually poor at assigning probabilities on a proposition.

In a system like Mycin, there is too much imperfect knowledge so that a rigorous probabilistic analysis is not possible. Shortliffe et al. devised their own theory of uncertainty, which was discussed before.

5.5. Problems with Shapiro's method.

We encountered a few problems when trying to use Shapiro's interpreter⁹³ for Propector. His basic interpreter looks like:

```
solve(true, []).
solve((A,B),[X|Y]) :- solve(A,X),solve(B,Y)
solve(A,F(S)) :- his_clause(A,B,F),solve(B,S).
```

We recall that F is called a certainty function, is monotone increasing and maps a list into some numeric value in $(0,1]$. In his paper, Ehud Shapiro defines clear semantics for logic programs with uncertainties. The question we asked ourselves was, how useful is it for existing expert systems?

One feature Propector and Mycin have in common is that all evidence for a particular hypothesis is collected before any conclusion is made. In Propector, for example, we saw that there are two ways of combining evidence:

1. logical combinations
2. multiple pieces of evidence

If a hypothesis has multiple pieces of evidence, each will influence the probability of the hypothesis independently of the other. Note that Prolog needs only one piece of evidence. On the other hand, the antecedent of an inference rule may also be a logical combination of evidences using the logical operators AND, OR and NOT. We note that Propector makes a distinction between

$$H - E_1 \quad (1)$$

$$H - E_2$$

and

$$H - E_1 \text{ or } E_2 \quad (2)$$

where Prolog does *not*.

In fact, pure Horn clause logic, on which Prolog is based, has no logical OR. In this theory, multiple clauses with the same consequent are used to denote OR. Prolog is one particular approach to Horn clauses. One could do a breadth-first search instead of a depth-first and still be logically consistent withing the framework of Horn clauses.

In a Prolog clause

$$H - E_1 ; E_2 \quad (2)$$

where ";" is the Prolog "or", the second argument E_2 gets evaluated only when the first one fails. In the multiple evidence situation however, all arguments of the premise part have to be evaluated. We would have to change the definition of ";" in Prolog to achieve this effect. This is one of the reasons why we decided that the most natural way to implement Prospector in Prolog was to take the meta-level approach. We also note that Mycin has no "or". Therefore, Mycin could be implemented on the object-level, i.e. in standard Prolog without having to rewrite the interpreter.

The reason a distinction is made between (1) and (2) in the case of Prospector is that (1) assumes conditional independence where (2) covers the cases where this assumption cannot hold because of interaction between the different pieces of evidence. We mean by this that, for example, the presence of one particular kind of ore deposit has its effect on the probability of the presence of another kind.

Thus, a problem occurs when trying to apply Shapiro's method to Prospector.

The question is how to deal with multiple evidence. Our solution was to change his interpreter into the following:

```
% we assume that all prior probabilities were defined
% by the domain expert who sets up the expert system
% the problem is to compute posterior probabilities

solve((A,true),V) :- solve(A,V).
solve((A,B),V) :- solve(A,V1),solve(B,V2),min(V1,V2,V).
solve((A;B),V) :- solve(A,V1),solve(B,V2),max(V1,V2,V).

% use of setof to handle multiple evidence
solve(A,V) :- rule_head(A),setof(B,clause(A,B),Bodies),
              solve_list(Bodies,List),compute(A,List,V).
solve(A,V) :- fact(A), % ask the user for his estimate or use the
                    % prior probability if he doesn't know

solve_list([],[]).
solve_list([H|T],[VH|VT]) :- solve(H,VH),solve(T,VT).
```

In the above program,

```
compute(Head,List,Value)
```

corresponds to formula (3).

This means that we are no longer within the semantically clean framework of logic programs with uncertainties as defined in Shapiro's paper. It is impossible to devise one certainty function that is monotone increasing, and that captures all of Propector's inference rules. This is due to the fact that this approach cannot deal with multiple evidence.

The major novelty brought by Shapiro's paper is that he shows that uncertainty can be dealt with at the meta-level, i.e. by extending the basic Prolog interpreter. However his method rests on fundamental assumptions that might constrain its applicability:

1. The certainty function has to fulfill some constraints, e.g. be monotone increasing. The question is whether inexact reasoning can be captured in such a certainty function. Shapiro doesn't make a point of it:

"Each school of expert systems is using its own particular way of

computing certainties, with no noticeable difference in the validity of their results. So we take our pick ..."⁹³

2. Another problem arises from assuming that an expert system may be modeled as a Horn-clause production system. Such a system must be modified to deal with multiple evidence. The interpreter in his paper⁹³ is based on the logic of Horn clauses: in order to prove a goal, you need to prove it only once. In real-world expert systems however, especially when dealing with uncertainties, all evidence has to be taken into account to conclude a hypothesis.

If Horn clauses would be suitably generalized, these problems could obviously be removed. However, redesigning Prolog was beyond the scope of this thesis. By using a meta-level approach, we obtained basically the same effect.

5.6. Prolog Implementation of Prospector

In this section, we will describe the main parts of a Prolog implementation of Prospector's reasoning strategy. The complete program and a trace of its execution are included in appendices one and two.

Our implementation is fairly complete. We implemented the *definition* and the *consultation phase*. We used the following strategy:

1. The prior probabilities of all propositions and inference rules in the given knowledge base are supplied by the domain expert and stored permanently.
2. The posterior probabilities of the pieces of evidence are supplied by the user and are stored temporarily.
3. All other posterior probabilities are computed by the inference system and are NOT stored at all.

The definition phase is the phase in which the knowledge engineer, in collaboration with the domain expert, defines the expert system.

The Definition phase

The program will ask the domain expert all necessary information needed. This consists of the following prior probabilities:

- a. For each rule $H :- E$:
 1. $P(H|E)$ which is stored in the program as `prior_phe(H,E,PHE)`;
 2. $P(H| \text{not } E)$ which is stored as `prior_phne(H,E,PHNE)`;
- b. For each rule head (hypothesis):

$P(H)$, i.e. the uncertainty of the hypothesis H is represented in the program as `prior_prob(H,P)`;
- c. For each piece of evidence E , i.e. non-rule head in knowledge base:

$P(E)$, i.e. the initial probability stored as `prior_prob(E,P)`;

The top level control of the definition phase looks like

```
ask_pro prospector_expert :-  
    print_expert,  
    clear_info, % remove all old information  
    ask_rules.
```

In this phase, the program collects all inference rules that constitute the knowledge base and asks for all necessary prior probabilities.

```
ask_rules :-  
    setof0(R,rule(R),Rules),  
    ask_info_rules(Rules).
```



```

ask_info_rules([]).
ask_info_rules([(X :- Y)|T]) :-
    nl,
    print_info,
    write(' What is the prior probability of '),
    write(X),
    write(' ? '),
    readnumber(Prob),
    assertz(prior_prob(X,Prob)),
    nl,
    write(' What is the prior probability of '),
    write(X),write(' if you know that '),
    write(Y),write(' is true'),
    write(' ? '),
    readnumber(PHE),
    assert(prior_phe(X,Y,PHE)), % P(H|E)
    nl,
    write(' What is the prior probability of '),
    write(X),write(' if you know that '),
    write(Y),write(' is false'),
    write(' ? '),
    readnumber(PHNE),
    assert(prior_phne(X,Y,PHNE)), % P(H|-E)
    ask_info_rules(T).

```

In our implementation, *fact* refers to *evidence*.

```

ask_facts :-
    setof(R,fact(R),Facts),
    ask_info_facts(Facts).

ask_info_facts([]).
ask_info_facts([X|T]) :-
    nl,
    print_info,
    write(' What is the prior probability of '),
    write(X),
    write(' ? '),
    readnumber(Prob),
    assertz(prior_prob(X,Prob)),
    ask_info_facts(T).

```

The Consultation Phase

Here, the program will be asking for posterior probabilities of the evidence. This means asking the user for his opinion about the various pieces of evidence. We let him provide $P(E_i' | E_i)$. This is a posterior probability. A fact, also called

a piece of evidence, is any statement (proposition) in the knowledge base, which is not a hypothesis for an inference rule.

The top level of the consultation phase looks like

```
ask_pro prospector_user :-
    print_user,
    setof(R, fact(R), Facts);
ask_user_info_facts(Facts).

ask_user_info_facts([]).
ask_user_info_facts([X|T]) :-
    nl,
    print_info,
    prior_prob(X,P),
    write(' The domain expert provided a probability of '),
    write(P), write(' for '), write(X), nl,
    write(' If you agree with this value'),
    write(' or you have no idea at all, then press CR '), nl,
    write(' otherwise enter your estimate '),
    readnumber(Prob),
    replace(Prob, post_prob(X,P)),
    ask_user_info_facts(T).
```

The following routine replaces old posterior probabilities of the pieces of evidence. When the user cannot supply any new evidence, then we use the prior probability, supplied by the domain expert in the definition phase.

```
replace(Prob, post_prob(X,P)) :-
    name(Prob, []),
    prior_prob(X,P),
    retract_check(post_prob(X,_)),
    assertz(post_prob(X,P)).

replace(Prob, post_prob(X,P)) :-
    retract_check(post_prob(X,P)),
    assertz(post_prob(X,Prob)).
```

The Prospector Subjective Updating Strategy

The routine `compute` forms the kernel of our implementation. As explained previously, it is basically an extension of the Prolog interpreter, written in Prolog. We called this the meta-level approach. `compute` takes as first argument a conjunction or disjunction of propositions and computes in its second argument the

posterior probability of the first. `compute` calls itself recursively.

A first case, to be considered, is an argument being the head of a rule, i.e. a conclusion or hypothesis with possible multiple evidence. Here `compute` will compute the posterior probability given the user's belief in the various pieces of evidence.

```
compute(Goal,Prob) :- rule_head(Goal),
                      setof0(Body,clause(Goal,Body),Bodies),
                      compute_likelihood(Goal,Bodies,L),
                      prior_prob(Goal,Prior),
                      calculate_odds(Prior,PriorOdds),
                      PostOdds is L * PriorOdds,
                      calculate_prob(PostOdds,Prob).
```

In case of a conjunctions or disjunctions of predicates, we use the basic fuzzy set formula.

```
compute((A,B),Prob) :- compute(A,P1), -
                       compute(B,P2),
                       min([P1,P2],Prob).

compute((A;B),Prob) :- compute(A,P1),
                       compute(B,P2),
                       max([P1,P2],Prob).
```

A negation of a proposition is treated using:

```
compute(not(A),Prob) :- compute(A,P1),
                        Prob is 1 - P1.
```

In case of a fact, we have to look up the user supplied estimate of the posterior probability

```
compute(A,Prob) :- evidence(A),post_prob(A,Prob).
```

We note that the meta-level definition `compute` corresponds to the forward propagation of uncertainty under a backward reasoning strategy. The most important routines, used by `compute`, are:

1. to compute the odds corresponding to a given probability:
`calculate_odds(Prob,Odds)`
2. to compute the probability corresponding to a given odds:

```
calculate_prob(Odds,Prob)
```

3. to compute the posterior probability of a hypothesis using Prosector's piecewise linear function:

```
compute_li(H,E,PostPHE)
```

4. to compute the effective likelihood ratio of $H|E$ in the case of multiple evidence:

```
compute_likelihood(G,[H|T],L)
```

The implementation of these routines is described in appendix 1.

A final note about our implementation of Prosector's reasoning strategy is that we used a special *consult* routine for loading a knowledge base. We wanted to store the database references of the Prolog clauses that constitute the knowledge base, for efficiency reasons. This enabled us to make a distinction between knowledge base and program. As *compute* is an interpreter for logic programs with uncertainties, we wanted it only to consider clauses that had a probability associated with them.

Our consult looks like:

```
consult2(File) :-
    seeing(Input),
    see(File),
    clear_knowledge_base,
    repeat,
    read(Term),
    process(Term),
    seen,
    see(Input),!.

process(Term) :- Term = end_of_file,!.
process(?-G)  :- !,call(G),!,fail.
process(Clause) :-
    assertz(Clause,R),
    assertz(ref(R)), % store the database reference
    fail.
```

By storing the database reference, we are able to make the distinction between program and data. We can also distinguish between facts and rules in the

following manner:

```
fact(F) :- ref(R),clause(F,true,R).  
fact(F) :- rule_body(B),and_member(F,B),not(rule_head(F)).  
  
rule(( X :- Y)) :- ref(R),clause(X,Y,R),Y == true.
```

The Execution of the Program

We ran our implementation on a Prospector inference net example taken from Gaschnig.⁴² We refer to appendices one and two for more details.

Conclusion

The meta-level approach provides a clean implementation: deductive reasoning is implemented using Prolog's basic deductive inference strategy and posterior probabilities are computed along the way. Throughout this work, we demonstrate that highly praised expert systems as Mycin and Prospector can be easily implemented in Prolog in a more natural and concise manner than the original LISP-based programs.

CHAPTER 6

BELIEF STRATEGY (MYCIN)

In this chapter, we will give a brief outline of the internals of the Mycin system,⁹⁴ and comment on its Prolog implementation. We refer to chapter three for the theoretical issues of Mycin's model of belief propagation.⁶⁴

6.1. Introduction

During the early meetings of the Mycin group, patient case histories were analyzed and an attempt was made to identify the criteria used by the medical experts in determining a therapy. It was decided that a computer program could be written using rules which encapsulate both the medical expert's knowledge and their decision process.⁹⁴ The aim of the program was to aid the doctor in determining a therapy for certain cases of bacterial infections of the blood. The program would be able to:

1. decide if a patient has a significant infection;
2. determine the likely identity of the organism that causes the infection;
3. decide on an effective treatment;
4. choose the most appropriate drug, given the patient's clinical condition.

In order to encourage doctors to accept and use the Mycin system, three further objectives were that the system should be

1. both useful and competent;
2. able to contain a large body of knowledge which is subject to frequent change;
3. able to communicate easily with its user and, in particular, be able to explain its decisions and decision making processes.⁹⁴

The system consists, as most expert systems, of the following modules:

1. A Consultation Module;
2. A Definition Module.

The consultation module handles the interaction with a user of the Mycin system. The definition module is to be used by the knowledge engineer, in cooperation with the medical expert, to add new medical knowledge to the knowledge base. Each of these modules uses two sub-modules

- a. Explanation Program;
- b. Question/Answering Program.

The explanation facility, one of the most important components of the system, serves both as a debugging facility for the knowledge engineer and as a help for the user.

There are also two conceptually different databases:

- a. Patient Database (case dependent knowledge);
- b. Knowledge Base (Mycin's rules)

The patient database is defined during consultations by the user; the knowledge base embodies the knowledge engineer's expertise, i.e medical knowledge on bacterial diseases.

6.2. The Consultation Subsystem

6.2.1. Data Structures

The most important part of the knowledge structure in Mycin is the set of some 450 rules which make up the knowledge base. Each rule contains both an action part and a premise. The action part contains a numeric quantity, called a *certainty factor*, which is a measure of the expert's confidence in the validity of the rule. This measure plays an important role in the inexact reasoning process, which will be described further.

When Mycin makes decisions, it takes a number of factors into consideration: the patient's clinical condition, his infections, the cultures taken from the patient, organisms which have been isolated from the cultures, and drugs which have been administered prior to the consultation. Each of these factors is called a *context*. Each time a new context is created, it is given a unique name and included in a *context tree*. In general, a context tree looks like

```

patient-1
  infection-1
    culture-1
      organism-1
      ..
    ..
  ..

```

The context tree structures the clinical problem and helps to infer relationships between contexts during the consultation.

Although the Mycin rules are responsible for the creation of context types in the context tree, the rules themselves are not linked at all in a decision tree or reasoning network. Each rule is modular and usually deals with one of the context types. The premise of a rule is, in general, of the form

```
( <predicate function> <obj> <attribute> <value> )
```

For example, the English-like rule

```

rule 009
  if 1. the stain of the organism is gramneg and
     2. the morphology of the organism is coccus
  then there is strongly suggestive evidence (0.8 )
     that the genus of the organism is neisseria

```

is represented in Mycin as

```

premise : (and (same cntxt gram gramneg)
              (same cntxt morph coccus))
action  : (conclude cntxt genus neisseria tally 0.8)

```

It is the purpose of (same cntxt gram gramneg) to cause the collection of all evidence bearing on the gramstain of the organism in question and then to find the certainty factor of the hypothesis, if any, which supports the possibility

that the gramstain is negative. If both clauses of the premise are valid, i.e. their certainty factors are greater than 0.2, then the deduction `genus is neisseria` is made with associated certainty factor `tally times 0.8`, where `tally` is the smaller of the two clause certainty factors. The threshold of 0.2 for certainty factors was determined by the Mycin designers.

6.2.2. Inexact Reasoning

Because of the uncertainty of medical diagnosis, an attribute or clinical parameter in Mycin always has a certainty factor (CF) which indicates a belief that the value of the attribute is the correct one.⁹⁷ The certainty factor may be deduced from the knowledge base or it may be supplied by the user in response to a question which demands from the user a value for the attribute. A certainty factor is stored as a real number in the interval `[-1,1]`, where positive values indicate evidence in support of the value and negative values suggest that the hypothesis is false. In practice, the user may provide an integer value in some predefined range which is then converted into `[-1,1]`.¹

In Mycin, values of attributes and their associated certainty factors are stored as 4-tuples. For example,

```
(site culture-1 blood-1.0)
 (ident organism-1 klebsiella 0.34)
```

We note that more than one hypothesis may be stored as evidence for the value of an attribute. In fact, the hypotheses for a particular attribute are stored together. A typical set of hypotheses would be stored as

```
val[organism-1,indent] = ((streptococcus 0.8)
                          (staphylococcus 0.4)
                          (dippneuminiae -0.7))
```

Predicate functions like `same` are used in the Mycin rules in almost all cases with respect to the set `val[C,P]`. This is the set of hypotheses which are relevant to the value of the clinical parameter `P` of the context type `C`.¹

We will illustrate the use of `val` by explaining the use of the function `same` in rule 009, described earlier. Let us suppose that the evidence accumulated for `organism1` is

```
val[organism1,gram] = (( gramneg 1.0 ))
val[organism1,morph] = (( rod 0.7 ) ( coccus 0.3 ))
```

Now, `same` returns the CF value supporting the hypothesis it is applied to, or NIL if no such evidence exists or the CF is 0.2 or less. Therefore,

```
( same organism1 gram gramneg )
```

returns the value 1.0 and

```
( same organism1 morph coccus )
```

returns 0.3.

To complete the evaluation of the example rule, we must explain "and." and differs from the usual Boolean conjunction in that it returns the value `true` only if each clause in the conjunction has a non-NIL CF of at least 0.2, a threshold determined by the medical expert. In the current example, the premise is evaluated as `true`, since the conjuncts have CF's 1.0 and 0.3 respectively, and the action statement would have been activated. In other words, the hypothesis (`neisseria 0.24`) would be added to `val[organism1,genus]`.

A more formal definition for `same` is

```
( same c p lst ) = cfmi if cfmi > 0.2
                  = nil  otherwise
```

where

```
cfmi = max { cf | (v,cf) in val[C,P] and v in lst }
```

We note that Mycin, unlike Prosector, has no logical "or." This made a meta-level approach unnecessary.

6.3. Explanations

The explanation capability of Mycin is one of the most important features for the medical user. It allows the user to obtain a much clearer picture of the reasoning processes involved in particular deductions or in the overall consultation. The three parts of the explanation facility provide the following features:

1. Displaying on demand, during the consultation, the rule which is currently being invoked.
2. The association, after the consultation, of specific rules with specific events and the explanation why each of them occurred.
3. Searching the knowledge base for a specific type of rule.

6.4. Prolog Implementation of Mycin

6.4.1. Introduction

In this section, we will give a detailed description of the Prolog implementation of Mycin. The complete program and a trace of its execution are included in appendices three and four at the end of this thesis.

Our design is similar to a program written in Waterloo Prolog by Peter Hammond of Imperial College, London in 1980.⁵¹ We used some parts of his program by converting them to C-Prolog. However, we extended the original design in various ways.

These extensions include :

1. Better I/O interface, and error recovery;
2. Modification of the explanation facility to make it more general;
3. Modification of the "dataclass" datatype to make the whole program more independent of its application area, i.e. a first step toward EMycin;

4. More meaningful names for predicates and variables;
5. Documentation of the program.

The data structures included in the Prolog version of Mycin are: a knowledge base in the form of rules; a context tree to store background data and important facts obtained during the consultation phase; and finally, a separate record of the possible identities of each suspected organism.

The context tree is defined by a datatype that reflects the hierarchical structure:

```
dataclass(class-number, class-name, class-details)
```

A record of possible identities of each organism is kept by

```
know(<organism>, <identity>, <certainty-factor>)
```

Finally, the predicate `rule` embodies the actual knowledge base.

6.4.2. The consultation system

The top level is initiated by `start_session` defined below.

```
start_session :- therapy_required, consultation.
start_session :- close_down.
```

The call `therapy_required` will ask the user if he has obtained positive cultures, so that the system will know if the consultation session should continue or not.

```
therapy_required :-
    print_header,
    get_answer(0, initiator, Answer), !,
    Answer = yes.
```

The call `get_data(0, PatientData)` starts the consultation phase.

```

consulation :-
    get_data(0, PatientData),
    nl,
    write(' The patient data obtained are:'),
    nl,
    write_out(PatientData).

```

The predicate `get_data` will ask for patient data in order to build a context tree.

```

get_data(Class1, [Class1DataItem|OtherClass1Data]) :-
    lt(Class1, 4),
    gen_new(Class1, Entity),
    get_details(Class1, Entity, Descr),
    sum(Class1, 1, Class2),
    get_data(Class2, Class2Data),
    Class1DataItem =.. [Entity, Descr, Class2Data],
    check_for_more(Class1, OtherClass1Data).

```

The end of the recursion in the data gathering process is reached by the call

```

get_data(4, Hypothesis) :-
    current(3, Entity),
    get_hypothesis(Entity, genus, Hypothesis).
% use the given knowledge base

```

The call `gen_new` will generate a new entity of the required class and also cause the printing of a headline announcing the new entity.

```

gen_new(Class, Entity) :-
    gen_no(Class, NewNo),
    dataclass(Class, ClassName, Details),
    concat(ClassName, NewNo, Entity),
    genheader(Entity).

```

`gen_no` generates a unique number for each new entity.

```

gen_no(Class,NewNo) :-
    retract(number(Class,OldNo)),
    sum(1,OldNo,NewNo),
    assert(number(Class,NewNo)).

genheader(Entity) :-
    write(' '),write(Entity),
    nl.

```

The procedure `get_details` asks the user to supply information of a background nature for a particular entity and can also cause a message to be printed, announcing the name of the first entity in the next class in the hierarchy.

```

get_details(Class,Entity,Descr) :-
    dataclass(Class,Name,Details),
    get_list(Class,Details,Descr),
    generate_messages(Class,Entity).

```

We note again that the datatype `dataclass` embodies the hierarchical structure of the context tree.

```

get_list(Class,[],[]).
get_list(Class,[Item|OtherItems],[Answer|OtherAnswer]) :-
    get_answer(Class,Item,Answer),
    get_list(Class,OtherItems,OtherAnswer).

```

The predicate `current` returns the current entity of a particular class

```

current(Class,Entity) :-
    number(Class,No),
    dataclass(Class,ClassName,Details),
    concat(ClassName,No,Entity).

```

`number` stores the number of a current entity for a particular class. This declaration serves as initial data for the `gen_no` predicate described earlier.

```

number(0,0).
number(1,0).
number(2,0).
number(3,0).
number(question,0).
number(rule,0).

```

The predicate `get_answer` will ask the user for data and read the value of an item of background data. Each question is preceded by a new question number.

```

get_answer(Class,Item,Answer) :-
    gen_no(question,Q),
    writeno(Class,Q),
    question(Item),
    readstr(Answer),nl.

```

`check_for_more` is a procedure to ask the user if there is another entity of a particular class to be considered.

```

check_for_more(Class,OtherData) :-
    dataclass(Class,ClassName,Details),
    write(' Is there another '),
    write(ClassName),write(' ? '),nl,
    write(' CR means no to me '),write('? '),
    readstr(Answer2),
    (Answer2 = unknown -> Answer = no;Answer = Answer2),
    nl,
    consider(Answer,Class,OtherData).

consider(no,Class,[]).
consider(yes,Class,OtherData) :-
    get_data(Class,OtherData).

```

The call `get_data`, in the previous procedure will continue the consultation process at any level in the context tree.

The rest of the predicates in this section deal explicitly with the predefined knowledge base. The predicate `same` is used in the rules that are described at the end of this section. Confidence factors are entered or defined on a scale of [-1000,+1000].

same will cause evidence to be gathered which bears on a particular value of a clinical parameter and succeeds if the confidence factor (CF) supporting this value is greater than 200, a threshold determined by the medical expert.

```
same(Entity,Attribute,RequiredValues,MaxCF) :-  
    get(Entity,Attribute,RequiredValues, MaxCF),!,  
    gt(MaxCF,200).
```

Note the use of "cut" in the above definition as we do not want the process of evidence collection to be repeated if the CF of the result is less than 200. same is defined in terms of get, a function that will compute the CF of the hypothesis, supported by Entity and Attribute.

The predicate get will collect the hypotheses relevant to determining the value of a clinical parameter and find the largest of the confidence factors supporting the possible values.

```
get(Entity,Attribute,RequiredValues,MaxCF) :-  
    get_hypothesis(Entity,Attribute,Hypotheses),  
    intersect(Hypotheses,RequiredValues,Intersection),  
    max_hypothesis(Intersection,MaxCF).
```

get_hypothesis is the most important procedure in this implementation of Mycin in Prolog. We have three possible methods of determining an attribute value:

1. First, we can test if it is already known in the current context

```
get_hypothesis(Entity,Attribute,[Hypothesis]) :-  
    know(Entity,Attribute,[Hypothesis]).
```

2. Next, we can try to deduce it from the knowledge base

```
get_hypothesis(Entity,Attribute,[Hypothesis]) :-  
    deduce(Entity,Attribute,[Hypothesis]).
```


3. Finally, we may ask the user to supply it. We note that the user always has the option to enter "unknown."

```
get_hypothesis(Entity,Attribute,[Hypothesis]) :-  
    ask_for(Entity,Attribute,[Hypothesis]).
```

We should also note that the real Mycin makes use of meta-rules in its definition of `get_hypothesis`. We did not include this powerful feature in our implementation, due to the small size of the knowledge base.

The predicate `deduce` collects all the evidence for the value of a parameter, merges evidence for the same value into one hypothesis and stores the information obtained in the context tree. By calling `rule-check` in a `setof`, `deduce` will inspect the whole knowledge base.

```
deduce(Entity,Attribute,Hypotheses) :-  
    setof(v(Value,CF),  
        rule_check(Entity,Attribute,Value,CF),Hypothesis1),  
        merge(Hypothesis1,Hypotheses),  
        assert(know(Entity,Attribute,Hypotheses))).
```

`rule_check` investigates a rule for applicability and calculates the CF of the deduction when the rule succeeds.

```
rule_check(Entity,Attribute,Value,CF) :-  
    is_rule(RuleNo),  
    asserta(currentrule(RuleNo)), % for explanations  
    rule(RuleNo,Entity,Attribute,Value,C,Tally),  
    product(C,Tally,CF).
```

The above call `rule(RuleNo,Entity,Attribute,Value,C,Tally)` will backtrack until an applicable rule is found.

We mentioned that the real Mycin system uses meta-rules to guide its search. These meta-rules are used to prune down the list of rules first fetched when evaluating a parameter. Using the meta-rules, different search strategies can be

implemented. Given the small sample knowledge base, we choose not to implement this feature. It could however easily be done using the meta-level approach, as we showed in the previous chapter on Prospector.

The procedure `merge` causes evidence for the same value of an attribute to be combined together to form a single hypothesis.

```
merge([], []).
merge([v(unknown, 1000)], []).
merge([H|Rest], [H1|R1]) :-
    compare_lists(H, Rest, H1, S1),
    merge(S1, R1).

compare_lists(R, [], R, []).
compare_lists(v(Value, CF1), [v(Value, CF2)|U], R, W) :-
    new_cf(CF1, CF2, CF3),
    compare_lists(v(Value, CF3), U, R, W).
compare_lists(v(Value1, CF1),
    [v(Value2, CF2)|U], R, [v(Value2, CF2)|W]) :-
    eq(Value1, Value2),
    compare_lists(v(Value1, CF1), U, R, W).

new_cf(CF1, CF2, CF3) :-
    diff(1000, CF1, CF4),
    product(CF4, CF2, CF5),
    sum(CF5, CF1, CF3).

intersect([v(Value, CF)|H], [Value|Rest], [v(Value, CF)|H1]) :-
    intersect(H, Rest, H1).
intersect([v(Value, CF)|H], [M|Rest], H1) :-
    ne(Value, M),
    intersect(H, [M], X),
    intersect([v(Value, CF)|H], Rest, Y),
    union(X, Y, H1).

intersect([], Anything, []).
intersect(Anything, [], []).

union([], Y, Y).
union(X, [], X).
union([R], Y, [R|Y]).
```

The predicate `max_hypothesis` computes the maximum of the confidence factors.

```

max_hypothesis([],0).
max_hypothesis([v(V,C)|H],MaxCF) :-
    max_hypothesis(H,C1),
    largest(C1,C,MaxCF).

largest(CF1,CF2,CF1) :-
    gt(CF1,CF2).
largest(CF1,CF2,CF2) :-
    gt(CF2,CF1).

```

ask_for causes the user to be asked for the value of a clinical parameter. The answer is read and a check is made to see if the answer is legal.

```

ask_for(Entity,Attribute,[v(ActualValue,CF)]) :-
    question(Entity,Attribute),
    parameter(Attribute,ExpectedValues),
    nl,
    writeln(' Please enter one of the following values: '),
    write(' '),
    write_list(ExpectedValues),
    write(' Default, CR, is the value unknown '),nl,
    read_answer(Answer1,CF1),
    check_for_query(Answer1,CF1,A,C),
    check_answer(Attribute,A,C,ActualValue,CF),
    assert(know(Entity,Attribute,[v(ActualValue,CF)])).

```

```

check_answer(Attribute,A1,C1,A1,C1) :-
    parameter(Attribute,ExpectedValues),
    member(A1,ExpectedValues).

```

```

check_answer(Attribute,A1,C1,A,C) :-
    parameter(Attribute,ExpectedValues),
    write(' Sorry, but we will have to do this part on the '),
    write(Attribute),
    write(' over again'),nl,
    write(' You will have to reenter the value and CF '),nl,
    writeln(' Please enter one of the following values: '),
    write(' '),
    write_list(ExpectedValues),
    read_answer(A2,C2),
    check_answer(Attribute,A2,C2,A,C).

```

```

question(Entity,Attribute) :-
    write(' Enter the '),
    write(Attribute),
    write(' of '),
    write(Entity).

```

`read_answer` is used to read to user's reply to a request for the value of a clinical parameter. The value and its certainty factor are both read (the default CF is 1000).

```

read_answer(Answer,CF) :-
    readstr(Answer),nl,
    write('Enter CF on scale [-1000,+1000], default 1000 (CR)'),
    nl,readstr(Something),
    ( Something = unknown -> CF = 1000 ; CF = Something ).

```

6.4.3. The Explanation System

The Mycin queries `why` and `rule` are easily implementable in Waterloo Prolog which has an "ancestor" system predicate. This predicate can be used to examine the ancestors of the literals which invoked the predicate. When "ancestor" is used with one argument, its argument is unified with the most recent ancestor for which this is possible. If the predicate succeeds and subsequently backtracking returns to this point in the proof, then the argument is unified with the next most recent ancestor, and so forth. This latter feature is most useful in the repeated use of `why`. As CProlog does not have this ancestor predicate, we used a global stack mechanism, implemented by `currentrule`, to implement the same capability.

The procedure `check_for_query` is called after the user is asked to give a parameter value. If either `why` or `rule` is input then the explanation system is used.

```

check_for_query(Answer,CF,A,C) :-
    member(Answer,[why,rule]),
    answer_query(Answer),
    get_nearest(Rule),
    report(Answer,Rule),
    check_again_for_query(A,C).

check_for_query(Answer,CF,A,C) :-
    A = Answer , C = CF. % i.e. no special action is taken

answer_query(why) :-
    write(' We are asking this to determine the genus '),
    nl.
answer_query(rule) :-
    writeln(' Current rule is ').

check_again_for_query(A,C) :-
    read_answer(A1,C1),!,
    test(A1,C1,A,C).

test(why,1000,A,C) :- fail.
test(A,C,A,C) :- eq(why,A).

get_nearest(rule(RuleNo,E,A,R,C,T)):-
    currentrule(RuleNo),
    rule(RuleNo,E,A,R,C,T).

report(why,Rule) :- explain(Rule).
report(rule,Rule) :- clause(Rule,Body),
    translate((Rule:-Body)).

```

explain prints the known parameters in a rule, those still to be determined and the deduction which should result.

```

explain(Head) :- clause(Head,Body),
                  divide(Body,Knowprems,Unknownprems),
                  write_known(Knowprems),
                  translate((Head :- Unknownprems)).

```

The next predicate investigates what evidence is known already, and what is still to be determined to come to a certain conclusion.

```

divide((A,B),(A,Otherknown),Unknown) :-
    known(A),
    divide(B,Otherknown,Unknown).
divide((A,B),[],(A,B)) :-
    known(A).
divide(A,[],A) :- !,known(A).

write_known([]).
write_known((A,B)) :-
    writeln(' It is know that : '),
    write_premise((A,B)),
    writeln(' therefore ').
write_premise([]).
write_premise((A,B)) :-
    translate_predicate(A),
    write_premise(B).
write_premise(A) :- translate_predicate(A).

```

The `translate` gives a simple English translation of a rule.

```

translate((Head :- Body)) :-
    Body = (!,min(_,_)),
    writeln(' we conclude : '),
    translate_predicate(Head).
translate((Head :- Body)) :-
    writeln(' if : '),
    write_premise(Body),
    writeln(' then : '),
    translate_predicate(Head).

```

```

translate_predicate(!). % no need to translate this
translate_predicate(min(M,N)). % no need to translate this
translate_predicate(same(E,A,R,C)) :-
    current(3,Entity),
    write(' the '),
    write(A),write(' of '),
    write(Entity),
    write(' is '),
    write_value(R),
    nl.
translate_predicate(rule(N,E,A,V,C,T)) :-
    current(3,Entity),
    write(' There is'),
    give_evidence(C),
    write(' evidence that the '),
    nl,
    write(' '),
    write(A),
    write(' of '),
    write(Entity),
    write(' is '),
    writeln(V),
    rule_no(N),
    nl.

```

know will determine if a piece of evidence is known already.

```

known(same(E,A,R,C)) :-
    know(E,A,Hypothesis),
    intersect(Hypothesis,R,I),
    max_hypothesis(I,MaxCF),
    gt(MaxCF,200).
known(!).

```

6.4.4. Data Structures

The `dataclass` predicate enables the user to declare the classes of data objects; their names; and the background details required. There are no "class-details" for "organism" because at this class level, we begin to use the knowledge base to obtain more specific information from the user, i.e. test each applicable rule.

```

dataclass(0,patient,[name,sex,age]).
dataclass(1,infection,[infection_type,infection_date]).
dataclass(2,culture,[culture_site,culture_date]).
dataclass(3,organism,[ ]).

```

We note that this definition of the data classes involved in the consultation process makes this process almost subject independent. Thus, using it for another application area should be straightforward.

The next important predicate in this Prolog implementation of Mycin is `parameter` which records the possible values of a clinical parameter.

```

parameter(genus,[unknown,strept,neisseria,bact,staph,coryn]).
parameter(gramstain,[unknown,pos,neg]).
parameter(morphology,[unknown,rod,coccus]).
parameter(conformation,[unknown,singles,longchains,shortchains]).
parameter(aerobicity,[unknown,anaerobic,facul]).

```

6.4.5. Knowledge Base

A typical Prolog version of a Mycin rule looks like

```

rule(9,Entity,genus,neisseria,800,Tally) :-
    same(Entity,gramstain,[neg],CF1),!,
    same(Entity,morphology,[coccus],CF2),!,
    min([CF1,CF2],Tally).

```

In the above rule, the call `same(Entity,gramstain,[neg],CF1)` causes all the evidence bearing on the gramstain of `Entity` to be collected. If the certainty factor of the hypothesis which suggests the gramstain is negative is larger than 200, then `CF1` takes this value. Otherwise the call `same` fails and the cut ("!") prevents the collection of evidence being repeated. If each clause in the premise succeeds, then `Tally` takes the value of the weakest of the certainty factors. We note here that the definition of `same` also contains a cut to avoid the repetition of unnecessary evidence collection. We refer to the program listing in the appendix

for other Mycin rules.

6.5. Conclusion

Throughout the various stages of development of both our implementation of Prospector and of Mycin, it has become clear that there are many benefits to be gained from using the logic programming language Prolog as the means for defining an implementing an expert system. The most obvious and most generally beneficial one is Prolog's inherent *clarity*. More specifically, the major advantages can be summarized as follows:

1. The incremental nature of constructing a Prolog program allows the expert system developer to determine the immediate effect of adding or deleting a rule.
2. There is a natural translation of a production rule representing an element of knowledge into a Prolog clause.
3. Prolog has a built-in search mechanism.
4. Unification provides pattern matching.
5. Backtracking can be used to find all possible paths in a search of a list of rules. There is no need for explicit programming of this behavior.
6. Prolog allows for quick and simple updating of the acquired knowledge during a consultation.

CHAPTER 7

COMPARISON OF PROBABILISTIC APPROACHES TO INEXACT REASONING

7.1. Introduction

In this chapter, we will put the major differences between the probabilistic systems treated so far on a row.

As described in previous chapters, we may classify the various probabilistic approaches into

1. Subjective Bayesian Reasoning (e.g. Prosector)
2. Measures of Belief and Disbelief (e.g. Mycin)
3. Dempster-Schafer Theory (e.g. VISIONS)

The fuzzy logic approach will be treated separately as it is not a strict probabilistic approach. The major differences between the different systems that we investigated are in:

1. The way in which uncertain information about knowledge is represented.
2. The assumptions that form the basis for propagating uncertainty.
3. The control structure used for this propagation.
4. The treatment, if any, of inconsistent information.

In this chapter, we will make a *cross-reference* between the three probabilistic approaches and the four major concepts in which they differ. Some of the material here is based on Quinlan's paper,⁸¹ who describes an alternative approach not treated here, and Cohen's dissertation.³⁰ This chapter may be regarded as a synopsis of most of the material that was described earlier. We also bundled our critique on the various probabilistic approaches together in this chapter. The basic structure of this chapter is based on Quinlan's paper.⁸¹

7.2 Knowledge Representation of Uncertainty

7.2.1. Subjective Bayesian Reasoning (e.g. Prospector)

a. Method

This approach provides a subjective, i.e. a user-defined, probability for each proposition. The adjective "subjective" means that the supplied probability measure does not come from the ideas of relative frequencies in the limit but can reflect a fair subjective estimate of the proposition being true.

b. Criticism

"First of all, a single-valued estimate tells us nothing about its precision, which may be very low when the value is derived from uncertain evidence".⁸¹

A second criticism, noted by Quinlan⁸¹ is that "a single value combines the evidence for and against a hypothesis, without indicating how much there is of each".

Another difficulty is that the Bayesian view of probability does not allow one to distinguish between *uncertainty* and *ignorance*. That is, one cannot tell whether a posterior probability or odds was directly calculated from evidence, or indirectly inferred from an absence of evidence.

7.2.2. Measures of Belief and Disbelief

a. Method

The validity of each proposition is expressed by two separate values. $MB(H,E)$ is a probability-like measure of belief in H given E and $MD(H,E)$ is a similar value for disbelief in H given E . The belief and disbelief measures are independent of each other and therefore cannot be probabilities. The two

measures are combined into a single assessment of H in the light of E , called the certainty factor $CF(H,E)$ and defined as

$$CF(H,E) = MB(H,E) - MD(H,E)$$

b. Criticism

Quinlan⁸¹ points out: "This two-valued approach is also subject to the same criticism of precision as Prospector-like systems, since both of the belief measures are also point values. It does overcome, however, the second objection because the interplay pro, $MB(H,E)$, and con, $MB(D,E)$, a particular hypothesis is clear". We note here that EMycin uses only certainty factors and is therefore, like Prospector, a single-valued system.

Mycin also suffers from the fact that certainty factors are sometimes interpreted as probabilities.

There is an interesting anecdote that applies both to Mycin and Prospector and that proves that both methodologies are pretty much *ad hoc*. Clancey and Cooper ran a study of Mycin's certainty factors in which they showed that "the rules use CFs that can be modified by $\pm .2$ without seriously affecting its advice."¹⁷ This brings us to a question raised by Davis:

"If the systems build around these mechanisms are too sensitive to the numbers produced, that's bad because no one is ready to defend the precise numbers. Yet if the systems are relatively insensitive to the numbers then all of that mechanism appears spurious."

7.2.3. The Dempster-Schafer Theory (e.g. VISIONS)

a. Method

“Instead of representing the probability of a proposition H as a point value, this approach bounds the probability of H to a subinterval $[s(A), p(A)]$ of the unit interval $[0,1]$. The exact probability $P(A)$ of A may be unknown but is bounded by

$$s(a) \leq P(A) \leq p(A)$$

Thus, the uncertainty about the precision of our knowledge about A is expressed as $p(A) - s(A)$. The above inequality can be reformulated as

- a. $P(A)$ is at least $s(A)$
- b. $P(\text{not } A)$ is at least $1 - p(A)$ ”⁸¹

b. Criticism

None at this point as the Dempster-Schafer theory deals with both previously mentioned criticisms of uncertainty representation.

7.3. Assumptions in the Inference

7.3.1. Subjective Bayesian Reasoning (e.g. Prosector)

a. Method

A problem arises when two distinct pieces of evidence E_1 and E_2 are relevant to a proposition H . Prosector overcomes this by assuming conditional independence:

$$P(E_1 | E_2 \text{ and } A) = P(E_1 | A)$$

$$P(E_1 | E_2 \text{ and not } A) = P(E_1 | \text{not } A)$$

b. Criticism

We quote from Quinlan:⁸¹ "This assumption received a lot of criticism,^{77,95} but a recent paper by Pearl⁷⁶ shows how this assumption can be maintained if propositions are generalized to multi-valued variables. If A , B and C are mutually exclusive and complete propositions, they can be replaced by a single proposition H with values A , B and C . The corresponding conditional independence assumption is thereby weakened, and independence of the pieces of evidence relevant to H is no longer implied".

7.3.2. Measures of Belief and Disbelief (e.g. Mycin)

a. Method

Mycin takes a similar approach to the above problem and uses the formula

$$MB(A, E1 \text{ and } E2) = MB(A, E1) + MB(A, E2) - MB(A, E1) \times MB(A, E2)$$

7.3.3. The Dempster-Schafer Theory (e.g. VISIONS)

a. Method

This approach also assumes conditional independence in the case of distinct pieces of evidence for the same hypothesis. We note here that for all three theories, making this assumption avoids the burden of asking the user for all joint probabilities.

Other basic assumptions in Prospector and Mycin include the fuzzy formulae to compute Boolean combinations of propositions and various approximations and

interpolation techniques in an attempt to correct for potential inconsistencies.

7.4. The Control Structure for Propagating Uncertainty

Quinlan⁸¹ observes: "Inference systems typically operate as consulting systems. They therefore draw a clear distinction between nodes representing propositions about which the system must be informed, i.e. evidence, and those that depend on others, i.e. hypotheses. This is true for both backward- and forward-chaining systems.

This means that the user can only provide input for propositions that cannot be inferred from others. The flow of inference is constrained to a single direction, from the propositions that constitute the "raw" evidence to the "goals." This "criticism" applies for all three approaches."

7.5. Dealing with Inconsistent Information

Neither the Bayesian nor the Mycin approach checks for inconsistency in the evidence for or against a proposition. It is only in systems such as the Dempster-Schafer approach, using an interval-valued probability, that there is a firm basis for detecting inconsistency in general. If the probability of a proposition A lies in the interval $[s(A), p(A)]$ and $s(A)$ is greater than $p(A)$, there is obviously something wrong.

However, although the latter approach is able to discover conflicts, the method does not provide any way to deal with it other than aborting the computation process. In the last chapter, we devoted a section on what we feel constitutes future work in this area. It is our belief that the use of *relaxation* or *constraint-propagation* techniques in inference systems would provide a natural mechanism to resolve conflicts.

0,1,2 and 3 are all elements of A , and 7,8,9 are not. But what about 4,5, and 6 ? Intuitively, 4 is more in A than 6 is, or more precisely, it is more plausible, i.e. *possible*, that 4 is an element of A than it is that 6 is an element of A . This notion of the *plausibility* of set membership, as opposed to the *probability* of set membership, leads to the generalization of the degree of membership in a set, and from this generalization comes *fuzzy set theory*.

A fuzzy set of some universe U is a collection of objects from U such that with each object is associated a degree of membership. The degree of membership is always a real number between 0 and 1, and it measures the extent to which an element is in a fuzzy set. In other words, it measures the plausibility of an element being in a particular set. A degree of membership of 0 corresponds to an element that is not in an ordinary set, and a degree of membership of 1 corresponds to an element which is in an ordinary set. For example, if we have a universe U being the set $\{a,b,c,d\}$, then a fuzzy subset, A , of this universe could be defined as

a is present in A with degree of membership 1.0
 b is present in A with degree of membership 0.9
 c is present in A with degree of membership 0.2
 d is present in A with degree of membership 0.0

Equivalently, A is written as

$$\{ 1/a , 0.9/b , 0.2/c \}$$

where the degree of membership is juxtaposed next to each element and elements of degree 0 are omitted.

The exact relationship of the notion of a fuzzy set to that of an ordinary set can be seen most clearly when one recalls the definition of the characteristic function of a set. For an ordinary set A , this function was defined as

$$char_A(x) : U \rightarrow \{0,1\}$$

3.2. Fuzzy Set Theory

While one can study this theory at a mathematically sophisticated level, it is also possible to gain a great deal of insight at a more introductory and expository level. At this more elementary level, one can consider fuzzy set theory to be a generalization of ordinary set theory: the theory of collections of things. In this theory, we can, for any set, A , define a function that determines for any element of the universe whether that element is a member of A . This function is called the *characteristic function* or *membership function* of A , and is defined by

$$\text{char}_A(x) = \begin{cases} 0 & \text{if } x \text{ is not in the set } A \\ 1 & \text{if } x \text{ is in the set } A \end{cases}$$

This function is defined for all the elements of the universe. It is a function mapping the whole of the universe U to the set of two elements $\{0,1\}$. We can write this as

$$\text{char}_A(x) : U \rightarrow \{0,1\}$$

With an identification of $\{0,1\}$ as $\{\text{true},\text{false}\}$, this characteristic function can also play a role in assigning truth values to statements about A . The most elementary statement about A is one of the form "x is an element of A ." In this case, the characteristic function also acts as a truth function: if x is an element of A , then $\text{char}_A(x) = 1 = \text{true}$.

Fuzzy set theory will basically generalize simple set theory by suitably modifying the notion of *membership* in a set. In traditional set theory, an element x is either completely *in*, or completely *out* of a set. Fuzzy set theory deals with half in and half out also. Consider the following example:

$$A = \{ x \mid x \text{ is a natural number and Mary's car can hold } x \text{ adult passengers} \}$$

Now suppose Mary's car is a Ford Pinto. Then it seems safe to state that

$$A \cap B = \{ \min(a(x), b(x)) / x \mid x \text{ is an element of } U \}$$

It has been shown that these definitions of the fuzzy union and the fuzzy intersection are natural and reasonable definitions extending the standard set theory notions of union and intersection.⁶³ The only other set operator from traditional set theory that is extended to fuzzy set theory is that of the *complement* of a set. The definition proposed by Zadeh for the complement of a fuzzy set A , from a universe U , is

$$\bar{A} = \{ (1 - a(x)) \mid x \text{ is in } U \}$$

As in ordinary set theory, the characteristic function of fuzzy sets links fuzzy set theory to fuzzy logic. The degree of membership of x in A corresponds to a "truth value" of the statement " x is a member of A ." This means that if S and T are statements in fuzzy logic, with truth values s and t , respectively, then the truth value of the statement " S and T " has to be $\min(s, t)$, corresponding to the definition of fuzzy intersection.

The fuzzy set operators presented so far are extensions of those from ordinary set theory. It is quite reasonable to expect that there will be important operations in fuzzy set theory that do not have their counterpart in ordinary set theory.

In fuzzy set theory, we can *concentrate* the fuzzy elements of a set by reducing the degree of membership of all elements that are only "partly" in a set and in such a way that the less the element is in the set, the more we reduce its membership. Another operation is to *dilate* a fuzzy set by increasing the membership of elements that are only barely in the set. *Normalizing* a fuzzy set is a process of adjusting the degree of membership of the elements so that at least one of them is "totally" in the set. We can *intensify* a fuzzy set by increasing the degree of membership of elements that are at least half in the set and decreasing the

where for a fuzzy set, it is

$$\text{char}_A(x) : U \rightarrow [0,1]$$

where here the degree of membership is the characteristic function.

The universe from which a fuzzy set is constructed need not be finite. Consider the following subset of the reals:

$$Y = \{ m(x)/x \mid x \text{ is a positive real number} \}$$

where

$$m(x) = \begin{cases} 1.0 & \text{for } 0 < x \leq 25 \\ \left[1 + \left(\frac{x-25}{5} \right)^2 \right]^{-1} & \text{for } x > 25 \end{cases}$$

The set Y can be thought of as a fuzzy set describing the imprecise term *young*, i.e. the set of ages of people who are young. Clearly, someone under twenty-five is young, so the degree of membership of a number less than twenty-five is 1.0. It is not so clear that a person who is thirty is young, e.g. a degree of membership of 0.5. In this way, the imprecision connected with the concept of youth can be captured mathematically and can be dealt with in an algorithmic fashion.

The definition of the basic operations on sets must also be modified for use in fuzzy set theory, and the most basic of these are *set union* and *set intersection*. The definition proposed by Zadeh was the following: if A and B are two fuzzy subsets of U and $a(x)$ is the degree of membership of x in A and if $b(x)$ is the degree of membership of x in B then

$$A \cup B = \{ \max(a(x), b(x))/x \mid x \text{ is an element of } U \}$$

and

$$m = \max_{x \in U} \{ a(x) \}$$

Normalization allows us to, in some sense, reduce all fuzzy sets to the same base. It insures us that at least one element of each fuzzy set will have a degree of membership equal to one.

Intensification

$$INT(A) = \{ m(x)/x \mid x \text{ is an element of } U \}$$

where

$$m(x) = \begin{cases} 2a^2(x) & \text{for } 0 \leq a(x) \leq 0.5 \\ 1 - 2(1 - a(x))^2 & \text{for } 0.5 \leq a(x) < 1.0 \end{cases}$$

Intensification acts like a combination of concentration and dilation. It raises the degree of membership of some elements, lowers some others, and modifies the steepness of the degree of membership curve. Since intensification increases the degree of membership only for the elements that have a degree greater than 0.5 and since it lowers the degree of membership of elements lower than 0.5, this operation heightens the contrast between the elements that are more than half in the set and those that are less than half.

Fuzzification

$$FUZ(A) = \bigcup \{ a(x) * K(x) \}$$

where K maps elements of U into fuzzy subsets of U , "*" is an extension of ordinary multiplication to the multiplication of a fuzzy set by a real number, and \bigcup is the fuzzy union discussed earlier. An example may clarify this operation. Let

degree of membership of the elements that are less than half in the set. Finally, another operation is to de-intensify or *fuzzify* a fuzzy set by increasing the extent of its fuzziness. All these inherently fuzzy operations are defined below for the fuzzy set $A = \{ a(x)/x \mid x \text{ is an element of } U \}$.

Concentration

$$CON(A) = \{ a^2(x)/x \mid x \text{ is an element of } U \}$$

The *CON* operator decreases the degree of membership of all elements, except for those with degrees of 0 or 1. In addition, it has the property that it decreases the membership of elements that have low degrees proportionally more than for elements with high degrees of membership.

Dilation

$$DIL(A) = \{ SQRT(a(x))/x \mid x \text{ is an element of } U \}$$

Dilation is the opposite of concentration. Elements that are barely in the set, e.g. with a degree of membership of 0.01, increase their degree of membership tremendously, e.g. tenfold in the case of a 0.01 degree of membership. All this follows, of course, from the fact that the square root operation is the inverse from the square operation, which also implies that

$$A = CON(DIL(A)) = DIL(CON(A))$$

Normalization

$$NORM(A) = \{ (a(x)/m)/x \mid x \text{ is an element of } U \}$$

where

8.3. The Concept of a Linguistic Variable

The very core of fuzzy set theory models an imprecise situation like the estimation of uncertainty by allowing one to estimate the plausibility or the possibility of an element being a member of a set. However, the theory can be difficult to use directly, especially for a novice. Fortunately, a *linguistic variable*, a notion build on top of fuzzy set theory, offers a viable alternative. The use of such a linguistic variable allows the precise modeling of imprecise statements like "LOW," "SOMEWHAT LOW" and so forth. Linguistic variables allow for the easy and natural specification of values for imprecise concepts, a specification that has a firm theoretical basis that can be performed behind the scenes.⁹⁸

A linguistic variable is a variable whose values are natural language expressions referring to some quantity of interest. These natural language expressions are then, in turn, names for the fuzzy sets composed of the possible numerical values that the quantity of interest can assume.¹¹¹

A detailed, small example of a linguistic variable may demonstrate its structure. Let us define one, named *Number*. The quantity of interest, i.e. our universe, will be an integer between 1 and 10. Let us assume that the set of natural language expressions that *Number* can take as its values is { *few* , *several* , *many* }. These values are names for the following fuzzy sets:

$$few = \{ .4/1, .8/2, 1/3, .4/4 \}$$

$$several = \{ .5/3, .8/4, 1/5, 1/6, .8/7, .5/8 \}$$

$$many = \{ .4/6, .6/7, .8/8, .9/9, 1/10 \}$$

The set of natural language expressions in which a linguistic variable takes its values is not an unrestricted set of English phrases. Rather it is a finite set that is structured by the system designer. This set can be specified using Backus-Naur Form (BNF), and incorporated in a logic grammar, such as a definite clause

$$U = \{ 1, 2, 3, 4 \}$$

$$A = \{ .8/1, .6/2 \}$$

$$K(1) = \{ 1/1, .4/2 \}$$

$$K(2) = \{ .4/1, 1/2, .4/3 \}$$

Then

$$\begin{aligned} FUZ(A) &= .8K(1) \cup .6K(2) \\ &= .8\{ 1/1, .4/2 \} \cup .6\{ .4/1, 1/2, .4/3 \} \\ &= \{ .8/1, .32/2 \} \cup \{ .24/1, .6/2, .24/3 \} \\ &= \{ .8/1, .6/2, .24/3 \} \end{aligned}$$

What we are doing in this operation is to fuzzify each element of a particular fuzzy set, e.g. the element 1 is mapped into 1 with a degree of membership 1 and into 2 with a degree of membership .4. This is the definition of $K(1)$ above, the fuzzification of the element 1. Analogously, $K(2)$ defines the fuzzification of the second element of the original fuzzy set. We note that the definition of fuzzification depends heavily upon the choice of the function K . Intuitively, we see that the operation of fuzzification has enlarged the set of elements with non-zero degrees of membership - the so-called *support* of a fuzzy set. More specifically, this support fuzzification spreads the degree of membership of individual elements of a fuzzy set to a possibly large number of elements, i.e. its support.

primary terms and hedges fulfill certain functional roles in the construction of the set of possible natural language expressions that a linguistic variable can assume as its values. The primary terms are the fundamental notions from which all the other elements of the set are built and the hedges allow for fine tuning of these primary terms.¹¹¹ When a system designer is building an expert system, using fuzzy logic, it is his goal to have a sufficiently rich set of primary sets and hedges so that the user feels almost unrestricted in his range of expressions and that he is able to associate with each possible natural language expression a technical, precise meaning that is still consistent with the imprecise English meaning. The expert system designer has three degrees of freedom: He selects the *primitives*, the *hedges*, and the *possible ranges* as well as their exact technical definitions.

The assignment of meaning to the primary terms is the assignment of a fuzzy restriction to each one of these terms. Suppose that we have a universe of the set of integers from 1 to 9. Then a possible fuzzy restriction that can be assigned to LOW is

$$\{ 1/1, .85/2, .53/3, .24/4, .08/5, .02/6 \}$$

The other primary terms, in the given grammar, can be assigned similar definitions. It cannot be overemphasized that these definitions are subjective, somewhat arbitrary choices made by the system designer *a priori*. It is up to him to make sure that these definitions are close to the user's intuition.

The *hedges* are not themselves modeled by fuzzy sets as the primary terms are, but rather are modeled as operators on the fuzzy restrictions that represent the primary terms. For example, in everyday English, the hedge "very" intensifies the particular word it modifies. An implementation of this hedge should therefore decrease the fuzziness of the elements of the fuzzy set that models the modified word. That is, the implementation of "VERY" should remove the elements that are only part of the way in the set, i.e. it should decrease the degree of membership for elements whose degree of membership is less than one. The

grammar. Below, we reproduce an example we took from Schmucker⁸⁹

<sentence> ::= <compound phrase> | <simple phrase>
<compound phrase> ::= <conjunctive phrase> | <range phrase>
<simple phrase> ::= <relational phrase> | <hedged primary>
<conjunctive phrase> ::= <relational phrase> AND <relational phrase>
<range phrase> ::= <hedged primary> TO <hedged primary>
<relational phrase> ::= <composite relation> THAN <hedged primary>
<composite primary> ::= <relation hedge> <relation> | <relation>
<relation hedge> ::= NOT | MUCH | SLIGHTLY
<relation> ::= LOWER | HIGHER
<hedged primary> ::= <hedge> <primary> | <primary> | <fuzzy number>
<hedge> ::= NOT | VERY | MORE OR LESS | FAIRLY | PRETTY | SORT OF |
EXTREMELY | INDEED | REALLY
<primary> ::= LOW | HIGH | MEDIUM
<fuzzy number> ::= <fuzzifier> <number>
<fuzzifier> ::= ABOUT
<number> ::= ONE | TWO | THREE | FOUR | FIVE | SIX | SEVEN | EIGHT |
NINE

This grammar covers a complex set of natural language expressions such as: "HIGH," "LOW," "MORE OR LESS HIGH," "ABOUT FOUR TO ABOUT SIX," "NOT LOW," "SLIGHTLY LOWER THAN PRETTY HIGH," and so forth.

The terms <hedge>, <primary>, <fuzzifier>, etc, play the role that "subject," "verb," etc., do in the construction of sets of English sentences, i.e., they are the building blocks used to construct the elements of the set. Of these, the *primary terms* and the *hedges* are the most important. Just as "verbs" and "subjects" have certain roles to fill in the construction of English sentences, so too the

concatenation of several hedges. An expression like "NOT VERY HIGH" can be represented by the fuzzy set obtained as follows: The operator *CON* is applied to the fuzzy restriction for *HIGH*, and then the operator *NOT* is applied to that result. Mathematically, we are determining the operator that represents a group of several hedges by the *functional composition* of the operators for the individual hedges. This has the advantages of ease of implementation and theoretical simplicity. For example, "VERY VERY LOW" can be represented by $CON(CON(LOW))$. This use of function composition automatically makes the expressions "NOT VERY HIGH" and "VERY NOT HIGH" different. The first is represented by $NOT(CON(HIGH))$ and the second by $CON(NOT(HIGH))$. These are not the same fuzzy set restriction and that is exactly what is desired from a thoughtful examination of the intuitive meanings of each natural language expression.

The negative side of representing hedges as operators is that some of them do not seem to be easily modeled by such an approach.

8.4. Prolog Implementation of Fuzzy Sets

The Prolog implementation of fuzzy set operators is included in appendix five. As fuzzy set theory is the underlying theory of fuzzy logic, this implementation will be used in our future work on implementing a fuzzy logic reasoning scheme in Prolog. The code is self explanatory and implements all the fuzzy set operators that were described in section 8.2.

8.5. Fuzzy Databases

concentration operator, *CON*, discussed in the previous section, performs this function and a reasonable implementation of the hedge "VERY" could be based on this operator in the following manner:

$$\text{VERY LOW} = \text{CON}(\text{LOW})$$

where "low" is the fuzzy restriction we have chosen for the primitive term LOW.

Other hedges may be defined in a similar manner. We remind the reader here that the definition of hedges are biased decisions of the expert system builder. The set of hedges that the system designer has to choose from is almost endless. As an example, we took some from Schmucker.⁸⁹

ESSENTIALLY	VIRTUALLY	VERY
SORT OF	RATHER	NOT
TECHNICALLY	ALMOST	MUCH
KIND OF	REGULAR	FAIRLY
ACTUALLY	MOSTLY	PRETTY
LOOSELY SPEAKING	IN ESSENCE	BARELY
STRICTLY	BASICALLY	REASONABLY
ROUGHLY	PRINCIPALLY	EXTREMELY
IN A SENSE	LOWER THAN	INDEED
RELATIVELY	HIGHER THAN	REALLY
PRACTICALLY	PARTICULARLY	MORE OR LESS
SOMEWHAT	LARGELY	PSEUDO-
EXCEPTIONALLY	FOR THE MOST PART	NOMINALLY
ANYTHING BUT	STRICTLY SPEAKING	LITERALLY
OFTEN	ESPECIALLY	TYPICALLY

Representing hedges as operators acting upon the representation of the primary terms has both positive and negative implications. On the positive side, it seems very natural and also allows for an easy implementation of the

8.5.2. Definitions

A general *fuzzy knowledge base* with a general deductive capability consists of base relations, virtual relations, set theoretic relations and functions.

Base relations are relations in which each tuple, i.e. row, satisfies the relation to some degree X which takes values in the interval $[0,1]$. Each column is associated with an attribute that can take values from an associated domain.

As an example, let us consider the following representation for *tall*:

<i>tall</i>	height	X
I-Type	5.7	0
	5.8	0.6
	5.9	0.8
	6.0	1
	7.0	1

A *virtual relation* is a relation defined by means of a rewrite rule in terms of base relations and other virtual relations, set theoretic relations and functions or queries. For example, let's assume we have the relation *tall* described above, together with a base relation *persons* defined in the following manner:

3.5.1. Introduction

In this section, we present *fuzzy databases*, i.e. databases that have the notion of fuzzy sets and/or fuzzy logic. We only consider so-called *deductive relational databases*, i.e. relational databases that have a deduction capability, such as Prolog²⁷ and DEDUCE.²³ A deductive database is essentially the same as a production system.

One problem with the existing database systems in the market today is that they only allow the user to ask very precise questions, i.e. non-fuzzy queries. For example, consider the query "*Find persons whose heights are greater than 5 feet 10 inches*". The criterion *5 feet 10 inches* is somewhat arbitrarily chosen and it will not find anybody who is, for example, 5 feet 9 inches. A better approach is to allow the user to ask a flexible query such as "*Find tall persons*". To build such systems, we inevitably come to the notion of fuzzy set.¹⁰⁶

Recent research^{7,24} has shown that fuzzy logic can be incorporated in deductive relational databases. Fuzziness can be dealt with at different levels:

1. Fuzzy Relations

A *fuzzy relation* is a relation where each tuple in the relation satisfies the relation to some degree X which takes values in the interval $[0,1]$. Thus a fuzzy relation is a fuzzy set.

2. Fuzzy Queries

A *fuzzy query* is a query containing a fuzzy concept. The result of a fuzzy query is a fuzzy set. A fuzzy query can operate on fuzzy and non-fuzzy relations.

Let us consider some base relations. The relation `likes` expresses the degree to which a person A likes a person B. Note that `likes` is not a symmetric relation.

```
likes(      jim,      irene,      1).
likes(      john,     heather,    0.7).
likes(      john,     mary,       0.6).
likes(      harry,    jill,       0.4).
likes(      jill,     tom,        0.2).
likes(      irene,    jim,        0.9).
likes(      heather, john,        0.8).
```

As our example will be dealing with the height of people, we have to define the fuzzy concept `tall`. In FRIL, this is an I-type base relation, i.e. a relation that takes linear interpolation values

```
tall( 5.25, 0).
tall( 5.50, 0.6).
tall( 5.75, 0.8).
tall( 6,    1).
tall( 7,    1).
```

Linear interpolation is between *consecutive* values. In the following code, we rely on the fact that the interpolation values are ordered in the base table.

```
tall(X,D) :-
    nonvar(X),var(D),not( clause(tall(X,_),true)),
    clause(tall(X2,Y2),true),nonvar(X2),X < X2,
    clause(tall(X1,Y1),true),nonvar(X1),X1 < X2, X1 < X,
    check(X1,X),
    D is ( ( (Y2-Y1)/(X2-X1) ) * ( X - X1 ) ) + Y1,!.

tall(X,D) :-
    nonvar(X), var(D), not( clause(tall(X,_),true)),
    clause(tall(X1,Y1),true),nonvar(X1),X < X1,
    D = Y1,!.


```

<i>persons</i>	name	height	weight	X
	jim	6.1	12.0	1
	john	5.9	11.9	1
	irene	5.5	10.0	1
	heather	5.6	9.6	1
	mary	5.3	8.5	1
	jill	5.7	9.2	1
	tom	6.0	13.5	1

then we could define a virtual relation *possible_athlete* being equivalent to tall persons and defined in the following manner

possible_athlete(X) :- persons(X,Y,_,_),tall(Y,I).

A *set theoretic relation* is defined by a procedure which takes as input a given tuple of values and returns a truth value in the interval [0,1]. For example, *less(X,Y)* takes two two values, i.e. numbers, and returns *true* is $X < Y$ and *false* otherwise.

An example of a *function* is *count* which takes the name of a relation as argument and returns the number of tuples in the relation.

In Prolog terms, a base relation corresponds to a unit clause and a virtual relation is basically the same as a nonunit clause.

8.5.3. Example of the Use of a Fuzzy Deductive Database

As an experiment, we implemented the language FRIL in Prolog. FRIL stands for Fuzzy Relational Inference Language and was designed by Baldwin.⁷ The complete code is included in appendix six. FRIL is a fuzzy relational database, i.e. each relation has an additional argument: the degree of membership, i.e. the possibility, of the tuple in the relation.

```
?- likes(X,Y,_),persons(Y,Z,_,_),tall(Z,1).
```

after which we get all the answers by backtracking:

```
Z = 6  
Y = tom  
X = jill ;  
  
Z = 6.1  
Y = jim  
X = irene ;  
  
no
```

Suppose we want to use a virtual relation in a query, an example of which is the question

Name those people with their weights who have a good friend

or in Prolog/FRIL:

```
! ?- has_good_friends(X),persons(X,_,Y,_).  
  
Y = 10  
X = irene ;  
  
no
```

We included a program script in appendix. Using the fuzzy set operators described earlier and their Prolog implementation, we can then implement queries such as.

Who is very tall ?

The hedge *very* corresponds to the fuzzy set operator *CON*, i.e. concentration.


```

tall(X,1) :- not( clause(tall(X,_),true)),tall(X,Y),!,Y =1,!.
check(X1,X) :- tall(Y,_),nonvar(Y),
              X1 < Y,
              Y < X,
              !,fail.

check(X1,X).

```

Another base relation, described earlier, is `persons` which contains data about persons. The Prolog version looks like

```

persons(      jim,      6.10, 12.00,1).
persons(      john,     5.90, 11.90,1).
persons(      irene,    5.50, 10.00,1).
persons(      heather,  5.60,  9.60,1).
persons(      mary,     5.30,  8.50,1).
persons(      jill,     5.70,  9.20,1).
persons(      tom,      6.00, 13.50,1).

```

With these base relations, we may define some virtual relations, such as

```

friends(X,Y) :- likes(X,Y,_),likes(Y,X,_).

```

which says that `X` and `Y` are friends if they like each other. Another virtual relation could be that tall persons are considered to be possible athletes or

```

possible_athlete(X) :- persons(X,Y,_,_),tall(Y,1).

```

A last example of a virtual relation is that a person has good friends if he has a possible athlete as friend.

```

has_good_friends(X) :- friends(X,Y),possible_athlete(Y).

```

With this given knowledge base, we may ask some questions that contain fuzzy concepts. We remind the reader that databases queries correspond to Prolog goals. Let us assume that, we want to know

Who likes a tall person ?

In our Prolog implementation of FRIL, this corresponds to

The greater the age, the older

The greater the height, the taller

A membership attribute may also be defined in terms of other attributes. We call such an attribute a *virtual* or *derived* membership attribute. For example

$$\text{size} = \text{height} * \text{weight}$$

the greater the size, the bigger

Once the knowledge of the membership attributes is coded into the knowledge base, we can use this to rank the answer of fuzzy queries instead of using the membership functions.

8.5.5. Interpretations of Fuzzy Queries

Let us assume that the fuzzy formula F is in conjunctive normal form, i.e. of the form

$$F = C_1 \& C_2 \& \dots C_r$$

where each of the C_i is a clause.

Let A and B be *atomic* fuzzy formulas. We assume that there exist membership attributes α and β for A and B respectively. Without loss of generality, we need to consider the following three cases:

1. F is the negation of A .

In this case, we sort the tuples by the values of the membership attribute α in ascending order.

2. F is the conjunction of A and B .

Here, we first sort all the tuples by values of the membership attribute α in the descending order. After the tuples are sorted, each tuple t is ranked by its position in the sorted list. let $r(t, \alpha)$ denote the *rank function* according to α . We

8.5.4. Acquisition of Fuzzy Knowledge

One of the difficulties for applying the fuzzy set theory to practical problems is to obtain *membership functions* of fuzzy sets. The degrees of membership of the elements of a fuzzy set are usually assumed to be known. Then, operations on fuzzy sets may be defined as we saw in the first section of this chapter. However, in reality it is not easy to obtain the precise membership functions, even for the atomic fuzzy sets. This is due to problems such as scaling, compatibility, and so on.

Fortunately, for evaluating a fuzzy query, we actually do not need to know membership functions themselves. We will explain this in a moment but first we have to make something clear. In the previous example, we represented the fuzzy concept *tall* as a fuzzy set, i.e. a fuzzy relation. In other words, the fuzzy concept was explicitly represented. It should be clear that we do not need to do it this way in order to be able to answer queries that contain the fuzzy concept *tall*. For example, Chang²⁴ does *not* store *tall*, but only *height*. The fuzzy concept *tall* is dealt with at the level of query processing. This is the approach to take when membership functions are unknown. What we need to know is some attributes such that a membership function is a *monotone increasing (decreasing)* function of these attributes. That is, find an attribute x such that the membership function f satisfies the following condition:

$$\text{if } x_1 > x_2 \text{ then } f(x_1) > f(x_2)$$

Such an attribute x is called a *membership attribute* of f . In the real world, finding a membership attribute is much easier than finding the membership function itself. For example, in our previous example, the attribute *height* is a membership attribute of the fuzzy concept *tall*.

To acquire fuzzy knowledge, e.g. to state the membership attributes, a system should allow the user to enter statements like these:

person A	person B	height of B	ranking
irene	jim	6.1	1
jill	tom	6.0	2
heather	john	5.9	3
harry	jill	5.7	4
john	heather	5.6	5
jim	irene	5.5	6
john	mary	5.3	7

8.6. Fuzzy Inference

In this section, we will give a basic outline of how fuzzy logic deals with inference rules. We first have to generalize the notion of fuzzy set and introduce some more definitions.

Definitions

As we have seen in section two, a fuzzy set A might have an infinite number of elements. The *support* of a fuzzy set A is the set of values in the universe U of which the degree of membership of the fuzzy set A , m_A is positive. A *crossover point* in A is an element of U whose grade of membership in A is 0.5. A *fuzzy singleton* is a fuzzy set whose support is a single point in U . If A is a fuzzy singleton whose support is the point y , we write

$$A = m/y$$

where m is the grade of membership of y in A . The symbol $=$ stands for "equal by definition" or "denotes." To be consistent with this notation, a nonfuzzy singleton will be denoted by $1/y$.

then sort in a similar manner all the tuples according to the other membership attribute, β , and obtain a rank function $r(t,\beta)$. Finally, we sort all tuples by the function

$$\max \{ r(t,\alpha) , r(t,\beta) \}$$

in ascending order.

3. F is the disjunction of A and B .

In this case, we first obtain the same rank functions $r(t,\alpha)$ and $r(t,\beta)$ as in the previous case and we sort according to the function

$$\min \{ r(t,\alpha) , r(t,\beta) \}$$

in ascending order.

The max and min functions correspond to the respective max and min functions used for the intersection and union of fuzzy sets.¹⁰⁶ The ordering of the tuples in the answer set is intended to help the user for his study. The user will still have the final judgement. To help him in this process, the system will print all the values of the membership attributes along with the tuples in the answer list.

For example, let us again consider our previous example with the base relations *likes* and *persons* but without the fuzzy relation *tall*. Assume that we have a membership attribute definition of the form

The greater the height, the taller

Then, using the methods described earlier, the answer to the fuzzy query

Who likes a tall person ?

will look like the following relation

Fuzzy Conditional Statements

A *fuzzy conditional statement* is a statement of the form

if A then B

in which A and B are fuzzy sets rather than propositions. A typical example might be

if x is small then y is large

In essence, this statement defines a relation between the fuzzy variables x and y . We first have to define the *cartesian product* of two fuzzy sets.

Let A be a fuzzy set in the universe of discourse U and B a fuzzy set in the universe V . Then, the cartesian product of A and B is denoted by $A \times B$ and defined by

$$A \times B = \int_{U \times V} \min(m_A(u), m_B(v)) / (u, v)$$

where $U \times V$ is the Cartesian product of the nonfuzzy sets U and V . The cartesian product $A \times B$ is a fuzzy set of ordered pairs (u, v) in $U \times V$ with the grade of membership of (u, v) given by $\min(m_A(u), m_B(v))$. In this sense, $A \times B$ is a fuzzy relation from U to V .

The significance of the fuzzy conditional statement of the form *if A then B* is made clearer by regarding it as a special case of the conditional statement *if A then B else C* where A , B and C are fuzzy subsets of possibly different universes U and V . In terms of the Cartesian product, the latter statement is defined as

$$\text{if } A \text{ then } B \text{ else } C = A \times B + (\text{not } A) \times C$$

A fuzzy set may be viewed as the union of its constituent singletons. On this basis, A may be represented as

$$A = \int_U m_A(y)/y$$

where the integral sign stands for the union of the fuzzy singletons $m_A(y)/y$. If A has a finite support $\{y_1, \dots, y_n\}$ then the above formula may be replaced by

$$A = \sum_{i=1}^n m_i/y_i$$

in which m_i is the grade of membership of y_i in A . It should be noted that the summation sign stands for union rather than the arithmetic sum.

A *fuzzy relation* R from a set X to a set Y is a fuzzy subset of the cartesian product $X \times Y$ and is defined by

$$R = \int_{X \times Y} m_R(x,y)/(x,y)$$

If the domains of X and Y are finite, then the relation R may be represented by a *relation matrix*.

If R is a relation from X to Y and S is a relation from Y to Z then the *composition* of R and S is a fuzzy relation denoted by $R \circ S$ and defined by

$$R \circ S = \int_{X \times Z} \max_y \left(\min \left(m_R(x,y), m_S(y,z) \right) \right) / (x,z)$$

If the domain of the variables x , y , and z are finite sets, then the relation matrix for $R \circ S$ is the max-min product of the relation matrices for R and S .

We also note that applications of fuzzy logic, e.g. in decision analysis and operations research, commonly use dynamic programming techniques to solve the max-min equations. In general, fuzzy sets that correspond to linguistic variables are defined by a function such as *young* in section two. This means that inference in fuzzy logic comes down to solving a linear or nonlinear program with local constraints on the variables.

8.7. Psychological Considerations of Fuzziness

As we described in the previous sections, the main advantage of using fuzzy logic to model uncertainty is that it captures the fuzziness that is contained in linguistic variables. One might wonder if we should allow for the input of natural language expressions as opposed for the input of numerical estimates such as probabilities. Are the complexities involved to deal with fuzzy natural language constructs worth the trouble? What are the gains of using fuzzy logic instead of a probabilistic approach to deal with uncertainty? These are the questions we answer in this section.

In Hofman and Neitzel,⁵⁴ the applicability of natural language input is tested for the specific domain of risk analysis. This domain is closely related to the area of expert systems. They conducted a preliminary study of the performance differences between subjects using natural language estimates and those using numerical estimates in the task of assessing the security risks of various computer installation configurations. Though this study used few subjects, it indicated that the use of natural language estimates rather than numerical estimates was associated with an increase of accuracy ranging from 16% to 32% due to the elimination of extremely inaccurate estimates. This study addresses only the effect of input *style*; however, it does not address the method by which these natural language expressions are modeled. Thus, with these indications that natural language results in dramatic improvements over numerical estimates, the importance of the

A typical problem encountered in a production rule is the following. We have a fuzzy relation, say R , from U to V which is defined by a fuzzy conditional statement. Then we are given a subset of U with typical element x and have to deduce a fuzzy subset of V with typical element y , which is induced in V by x . For example, we might have the following two statements:

1. x is very small
2. if x is small then y is large else y is not very large

of which the second defines a fuzzy relation R . The question is then: "What will be the value of y if x is very small? The answer to this question is provided by the following rule, which may be regarded as an extension of the familiar rule of *modus ponens*.

The *compositional rule of inference* says that if R is a fuzzy relation from U to V and x is a typical element of fuzzy subset of U , then the fuzzy subset y of V which is induced by x is given by the composition of R and x , that is

$$y = x \circ R$$

in which x plays the role of a unary relation. Thus, for finite x , y and R , y is defined as the max-min product of x and R . We note the analogy of the compositional rule of inference with Bayes rule for conditional probabilities. We should also note that because of the min-max product, the relation between x and y is not continuous.

As a final comment, it is important to realize that in practical applications of fuzzy conditional statement to the description of complex or ill-defined relations, the computations involved would in general be performed in a highly approximate fashion. Furthermore, an additional source of imprecision would be the result of representing a linguistic variable, e.g. *small*, by a fuzzy set.

The theory of approximate reasoning is made up of two components, representation or translation rules and inference rules. The representation rules provide a procedure for translating natural language statements into possibility distributions and thus providing a quantitative representation of the information contained in a statement. The rules of inference provide a means for manipulating information, represented as possibility distributions, to get new information. This new information is also represented by a possibility distribution which can then be retranslated into an atomic proposition.

We showed in this chapter that fuzzy logic is a much richer tool to deal with uncertainty in expert systems. We are, however, aware of some issues:

1. The knowledge acquisition process will be much more time-consuming than under a probabilistic strategy. The reason is that, by definition, the fuzzy logic approach, requires the expert and/or knowledge engineer to clearly define what he means.
2. Fuzzy logic is complex, i.e. almost as complex as human reasoning under uncertainty. Up till now, fuzzy logic was a sub-field of mathematics. Recently, however, we have witnessed introduction of this theory in the area of Artificial Intelligence.^{46,55,5,47,6}

We do believe, as natural language and/or expert systems will become common place, that the need for fuzzy logic will become more apparent.

behind-the-scenes modeling of the natural language terms is greatly increased.

The use of fuzzy sets in this behind-the-scenes modeling to directly represent the primitives and to indirectly represent the hedges of natural language expressions is in accordance with the psychological studies of the way people use such expressions. It has been shown that people naturally form fuzzy sets when classifying objects.^{69,72} By saying that people form fuzzy sets when dealing with vague concepts, we mean that the way people manipulate these concepts is very much like the operators we have defined for fuzzy sets. However, while people comprehend vague concepts *as if* the concepts are internally represented as fuzzy sets, they do not always manipulate these concepts in *exactly* these ways as we have defined the fuzzy set operators. The choice and definition of an operator is always a matter of context, mainly depending on the real world situation that is to be modeled.⁴⁰ In other words, all mathematical properties regarding the class of fuzzy set operators must be interpreted at an intuitive level. Experiments have demonstrated that fuzzy set theory is adequate to model the fuzziness of real-world situations. The correspondence between linguistic hedges and fuzzy set operators has been experimentally verified. What remains as a "disagreement" between mathematics and psychology is the *exact form* these operators should take.⁸⁹

8.3. Conclusion

Much of the information available in the real world involves linguistic information. Furthermore, much of the reasoning performed in practical problems involves inferences with this linguistic information. The theory of fuzzy sets introduced by Zadeh and developed by numerous researchers provides a mechanism for representing information of this imprecise type. The theory of fuzzy logic provides a method, based on the concept of fuzzy sets, for both representing and reasoning with imprecise concepts.

engineering to be based on linguistic reasoning, and not the artificially quantitative approaches that form the stereotype in modern science. He has developed a complete system of fuzzy logic that allows imprecise, but practically valid arguments in everyday reasoning to be translated in computable terms. This is highly needed and justified. Just consider executive management, for example, where one of the key aspects is planning under uncertainty. Normal language provides a means for imprecision to be clearly and exactly expressed. The current natural language front-ends to databases such as LIFER⁵³ are highly restricted, not in the quantity of information, but in its quality and richness, compared with that used in comparable human reasoning within an organization.

It is well known that there are traditionally two camps in the Artificial Intelligence community:

1. Those who believe knowledge representation should be based on logic (first-order logic, modal logic, ..). Examples are McCarthy, Nilsson, Bibel and Kowalski.
2. The people who think representation of knowledge should be based on ad hoc methods and heuristics. Examples are Schank and Minsky.

Recently, we have seen the birth of a third camp, closely related to the first: the fuzzy logic people. They believe in fuzzy logic as the logic underlying inexact reasoning.

The traditional approaches to reasoning under uncertainty are based on the premise that probability theory provides the necessary and sufficient tools for dealing with the uncertainty and imprecision. The theory of fuzzy sets calls into question the validity of this premise. More specifically, it suggests that much of the uncertainty which is intrinsic in knowledge-based systems is rooted in the fuzziness of the information, which resides in the database, and in the fuzziness of the underlying probabilities. Viewed in this perspective, then, it is the failure of the classical probability theory to come to grips with the issue of fuzziness of data

CHAPTER 9

CONCLUSION AND FUTURE WORK

In this final chapter, we briefly review the motivation for this thesis work, our major accomplishments and directions for future research suggested by this work.

9.1. Motivation for this Work

This exploration into the world of inexact reasoning originated from our interest in expert systems technology and logic programming. The growing set of literature on expert systems and their current commercialization fails to mention the unsolved issues of this field. One of the most important of these is, in our opinion, the modeling of reasoning under uncertainty.

The engineering approach to uncertainty is simply when formulating a problem to avoid the representation of uncertainty. This corresponds to the *Closed World Assumption*, taken by Prolog, which is used to delimit what a system must know.⁸⁵ The Closed World Assumption says: "If x is true, then it is deducible from the database." Under the Closed World Assumption, one assumes that something is not true if it cannot be shown that it is true, fails to show that it is true. This has been called *Negation as Failure*.²⁵ This stands in contrast with one of the most important aims of Artificial Intelligence: "trying to model human reasoning which is inherently non exact." Let us think of databases first. Adding a fuzzy logic component would highly extend their power: one would be able to talk about concepts that are partly in a relation, one would be able to deal with queries that contain fuzzy concepts, etc.. We touched on this in the sections of fuzzy databases.

Database theory, even though it allows for non-numerical data, still requires these data to be precise and well-defined. However, databases are moving out of the exact domains. Zadeh has argued for many years the need for systems

In this thesis, we explored the different methodologies for inexact reasoning in expert systems. It is well known that real world expert systems have to be able to cope with uncertainty to make them somehow "intelligent." Expert systems should, by definition, model the reasoning process of an expert in a particular application domain. Most commonly however, experts have to make decisions under conditions of imperfect information and uncertainty. Artificial intelligence researchers were already aware of this problem since the early days of expert systems. Various methods to describe this inexact reasoning process were proposed and some of them were implemented, with moderate success.

We first investigated the different probabilistic approaches that are currently used. These methods are all based on some notion of probability and they form the most widely used strategy. As an experiment, Prospector-like and a Mycin-like inexact reasoning schemes were implemented as a Prolog program. Prospector is an expert system for mineral exploration and is one of the most successful commercial applications of artificial intelligence technology. Mycin is an expert in bacterial diseases.

As observed by many researchers in this field, none of the existing methods for reasoning under uncertainty is completely satisfactory.

All our criticism on probabilistic approaches has been concentrated in chapter seven. The main problem, each of the existing systems has, is being unable to guarantee internal consistency.

We gave an introduction to fuzzy logic and demonstrated the reasoning power this theory could bring to expert systems, especially those with which the user communicates in a natural language.

that limits its effectiveness in dealing with a wide variety of problem areas, including expert systems, in which some of the principal sources of uncertainty are not statistical in nature.

In applying the theory of fuzzy sets to the analysis of real-world problems, it is natural to adopt the view that imprecision in primary data should, in general, induce imprecision of the same proportion in the results of the analysis. It is basically this view that motivated Zadeh the introduction of a linguistic variable, that is, a variable whose values are not numbers but words or sentences in a natural or synthetic language.¹¹¹ The theory of fuzzy sets provides a framework for dealing with such variables in systematic and consistent way and thereby opens the door to the application of the linguistic approach in a wide variety of problem areas.

9.2. Conclusion

Thesis

The two initial goals of this research were:

1. Augment Prolog with an inexact reasoning component.
2. Investigate the various methods for dealing with uncertainty in a critical manner.

Summary

Logic programming languages, notably Prolog, have features that make them natural for pattern-oriented programming. These languages offer simplicity and ease of use, the ability to express both factual and procedural knowledge declaratively, backtracking facilities and a restricted theorem proving capability based on Horn clauses and resolution. Prolog has also received special attention since its selection by the Japanese as a cornerstone of their Fifth Generation Project.

efficient implementation. We would like to explore a relaxation approach as implementation methodology. This would provide us with a lot of insight in the problems of both relaxation in logic programming and the validity of fuzzy logic in modeling inexact reasoning.

Relaxation is a general computational paradigm for finding solutions for a given set of equations that satisfy a number of constraints.⁴⁵ It has applications in interactive graphics, solution of constrained search problems, and efficient execution of recursively defined pattern matching procedures. Relaxation primitives in a logic programming language will give both naive and experienced users significant new power, and will be particularly useful for writing expert systems or production systems in which programming is done through pattern matching and application of rules.

We would like to explore the use of relaxation in probabilistic schemes.⁸⁰ As an example application in expert systems, one may want to specify measures of confidence on certain events using Bayesian assertions on probabilities. For example, let us view the set of possible events as a graph as in Duda et al.⁴¹ and each event is a node in the graph and edges connect related events. If we model the problem as one of labeling nodes, and let $p_i(\lambda)$ be the probability of label λ at node i , where $r_{ij}(\lambda, \lambda')$ is a subjective measure of the compatibility of label λ at node i and label λ' at the neighboring node j , then a homogeneous Bayes rule for calculating p in terms of r is

$$p_i(\lambda) = \frac{q_i(\lambda)}{\sum_{\lambda} q_i(\lambda)}$$

where $q_i(\lambda)$ is defined by

$$q_i(\lambda) = p_i(\lambda) \prod_{j, \lambda'} r_{ij}(\lambda, \lambda') p_j(\lambda')$$

Results

In this thesis, we demonstrated two fundamental things:

1. Fuzzy logic is more adequate than the various probabilistic approaches to model human reasoning because it captures more uncertainty. Probabilistic approaches basically come down to first-order logic plus probabilities. Fuzzy logic extends multi-valued logic with fuzzy quantifiers and fuzzy probabilities.
2. It has been said that, as Prolog is a production system, it is an ideal language for goal-driven expert systems. However, one component not readily available in Prolog is a fuzzy reasoning strategy. In this thesis, we showed how one could implement probabilistic approaches using either a meta-level approach (cleaner) or an object-level approach. We also showed how a non-probabilistic method, fuzzy logic, can be integrated into Prolog in a consistent manner.

9.4. Future Work

This research suggested some new directions that are worthy to investigate.

We would like to explore the idea of using *relaxation* techniques to handle fuzziness in production systems. The first major effort would be to use relaxation to implement fuzzy logic.

Fuzzy logic is a well-defined, mathematically sound and complete formalism to deal with uncertainty that captures more of the real world fuzziness than traditional probabilistic strategies. In particular, it deals with the vagueness, inherent to natural language concepts. Fuzzy logic has well-defined notions of internal consistency, unlike the more traditional approaches. We also note here that fuzzy logic up till now has not been used in all its aspects in existing expert systems. Most of the research has been theoretical and there seems to be a problem of

BIBLIOGRAPHY

1. Adams, J.B., "A Probability Model of Medical Reasoning and the MYCIN Model," *Mathematical Biosciences* 32 (1976).
2. Aikins, J.S., J.C. Kunz, H. Shortliffe, and R.J. Fallat, "PUFF: An Expert System for Interpretation of Pulmonary Function Data," STAN-CS-82-931, Stanford University Computer Science Dept (1982).
3. Anderson, S.O., R.C. Backhouse, J.C. Neves, and M.H. Williams, "A Prolog Implementation of Query-By-Example," Technical Report, Heriot-Watt University, Edinburg (1982). Also in Proceedings of the Seventh International Symposium on Computing, Germany
4. Baldwin, J.F., "Fuzzy Logic and Fuzzy Reasoning," in *Fuzzy Reasoning and Its Applications*, ed. E.H. Mamdani and A B.R. Gaines, Academic Press (1981).
5. Baldwin, J.F. and B.W. Pilsworth, "An Inferential Fuzzy Logic Knowledge Base," Technical Report, Department of Engineering and Mathematics, University of Bristol, U.K. (1982).
6. Baldwin, J.F., "Fuzzy Expert Systems," *Proceedings International Symposium on Multi-Valued Logic*, Japan (1983).
7. Baldwin, J.F., "Knowledge Engineering Using a Fuzzy Relational Inference Language," *Proceedings of the Conference of the International Federation on Automation and Control*, Marseille, France (1983).
8. Barnett, J.A., "Computational Methods for a Mathematical Theory of Evidence," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver, Canada, IJCAI-81 (1981).
9. Barrow, H.G., "Proving the Correctness of Digital Hardware Designs," *Proceedings of the National Conference on Artificial Intelligence*, Washington D.C., AAAI-83 (August, 1983).
10. Battani, G. and H. Meloni, "Interpreteur du Langage de Programmation Prolog," Technical Report, Groupe d'Intelligence Artificielle, Universite d'Aix-Marseille II, Marseille, France (1973).
11. Bayes, T., "An Essay Towards Solving a Problem in the Doctrine of Changes.," *Philosophical Transactions of the Royal Society* 53, pp. 370-418 (1763).
12. Eclovari, G. and J.A. Campbell, "Generating Contours of Integration: An Application of Prolog in Symbolic Computing," pp. 14-23 in *Lecture Notes in*

Such a requirement corresponds to a set of constraints on the current confidence measures of an event $p_i(\lambda)$, linking it to the measures for its antecedents and consequents in the graph.

We also feel that fuzzy logic could be implemented more efficiently in a logic programming language that has a relaxation-based search mechanism to replace the exhaustive backtracking of standard Prolog.

Ideally users should be able to present a computer with problems directly, and await responses containing solutions. Existing programming systems are compromises towards this ideal, compromises that cope poorly with many users' problems. Many of these problems consist of sets of constraints, for which users seek feasible solutions. Therefore, adding a constraint satisfaction scheme or relaxation to Prolog-based expert systems is a worthwhile endeavor.

26. Clark, K.L. and F.G. McCabe, "Prolog: A Language for Implementing Expert Systems," *Machine Intelligence*(10) (1982).
27. Clocksin, W.F. and Chris S. Mellish, *Programming in Prolog*, Springer Verlag, New York (1981).
28. Clocksin, W.F. and J.D. Young, "Prolog," *Computervorld* 17(31) (August, 1983).
29. Coelho, Helder, "Database Interrogation by Means of Natural Language," *Proceedings the First International Workshop on Natural Language Communication with Computers*, Warsaw, Polen, Held in Warsaw, Polen (September, 1980).
30. Coelho, Helder, "A Formalism for the Structural Analysis of Dialogues," *Proceedings of the 9th International Conference on Computational Linguistics* (July, 1982).
31. Cohen, Paul R., "Heuristic Reasoning about Uncertainty: An Artificial Intelligence Approach," Technical Report STAN-CS-83-986, Stanford University, Department of Computer Science (August 1983).
32. Colmerauer, Alain, "An Interesting Subset of Natural Language," in *Logic Programming*, ed. K.L. Clark and S.-A. Tarnlund, Academic Press (1982). A.P.I.C. Studies in Data Processing No. 16
33. Dahl, Veronica, "Logical Design of Deductive Natural Language Consultable Data Bases," *Proceedings Fifth International Conference on Very Large Data Bases*, Rio De Janeiro, Brazil (1979).
34. Dahl, Veronica, "On Database Systems Development Through Logic," *ACM Transactions on Database Systems* 7(1), pp. 102-123 (March 1982).
35. Darvas, F., I. Futo, and P. Szeredi, "A Logic Based Program System for Predicting Drug Interactions," *International Journal of Biomedical Computing* 9(4) (1977).
36. Davis, R., "Panel on Dealing with Uncertainty," *Proceedings of the 6th International Joint Conference on Artificial Intelligence*, pp. 1101-1102 (1979).
37. Dempster, A.P., "Upper and Lower Probabilities Induced by Multivalued Mapping," *Annals of Mathematical Statistics* 38, pp. 325-339 (1967).
38. Dempster, A.P., "A Generalization of Bayesian Inference," *Journal of the Royal Statistical Society* 30, pp. 205-247 (1968).
39. Dinbas, M., "A Knowledge-Based Expert System for Automatic Analysis and Synthesis in CAD," *Proceedings IFIP 80*, AFIPS Press (1980).
40. Dubois, D.J. and H. Prade, "Operations in a Fuzzy-Valued Logic," *Information and Control* 43(2), pp. 224-240 (November, 1979).
41. Duda, R.O., P. Hart, and N. Nilsson, "Subjective Bayesian Methods for Rule-

- Computer Science 87: 5th Conference on Automated Deduction*, ed. Wolfgang Bibel and Robert A. Kowalski, Springer-Verlag, Berlin, Germany (1980).
13. Bijl, A., Fernando C.N. Pereira, and P.S.G. Swinson, "A Fact Dependency System for the Logic Programmer," *Computer-Aided Design*, Also available as EdCaad Technical Report 82-03, University of Edinburgh (July, 1983).
 14. Borning, A., "A Powerful Matcher for Algebraic Equation Solving," Working Paper 67, Department of Artificial Intelligence, University of Edinburgh, Scotland (May, 1980).
 15. Borning, A. and Alan Bundy, "Using Matching in Algebraic Equation Solving," *Proceedings of the International Conference on Artificial Intelligence 1981*, pp. 466-471, Also available from The University of Edinburgh as DAI Research Paper No. 158 (1981).
 16. Bowen, K.A., "Logic Programming and Relational Databases," in *Workshop on Logic Programming*, ed. Sten-Ake Tarnlund, Debrecen, Hungary (July 1980).
 17. Brooks, A., "A Comparison Among Four Packages for Knowledge-Based Systems," *Proceedings of the International Conference on Cybernetics and Society*, pp. 279-283 (1981).
 18. Buchanan, Bruce and Edward Shortliffe, *Building Expert Systems with Production Rules: The MYCIN Experiments*, Addison-Wesley (1984).
 19. Bundy, Alan, Lawrence Byrd, G. Luger, Chris Mellish, and M. Palmer, "Solving Mechanics Problems Using Meta-Level Inference," pp. 153-167 in *Expert Systems in the Microelectronic Age*, ed. D. Michie, University of Edinburgh, Scotland (1979).
 20. Bundy, Alan and L.S. Sterling, "Meta-Level Inference in Algebra," Research Paper 164, Department of Artificial Intelligence, University of Edinburgh, Scotland (September, 1981). Presented at the Workshop on Logic Programming for Intelligent Systems, Long Beach, California, 1981
 21. Bundy, Alan, *The Computer Modelling of Mathematical Reasoning*, Academic Press (1983).
 22. Carnap, R., "The Two Concepts of Probability," pp. 19-51 in *Logical Foundations of Probability*, University of Chicago Press, Chicago (1950).
 23. Chang, C.L., "DEDUCE 2: Further Investigations of Deduction in Relational Data Bases," in *Logic and Databases*, ed. H. Gallaire and J. Minker, Plenum Press (1978).
 24. Chang, C.L., "Decision Support in an Imperfect World," Research Report RJ3421, IBM Research Division, San Jose, California (March, 1982).
 25. Clark, K.L., "Negation as Failure," in *Logic and Databases*, ed. H. Gallaire, J. Minker, Plenum Press (1978).

- IEEE 1980 International Conference on Cybernetics and Society*, Boston, Massachusetts (October, 1980).
55. Horn, A., "On Sentences Which are True of Direct Unions of Algebras," *Journal of Symbolic Logic*(16) (1951).
 56. Ishizuka, M., K.S. Fu, and J.T.P. Yao, "A Rule-Based Inference System with Fuzzy Sets for Structural Damage Assessment," in *Fuzzy Information and Decision Processes*, ed. M. Gupta and E. Sanchez, North-Holland, Amsterdam (1982).
 57. Ishizuka, M., "Inference Methods Based on Extended Dempster and Shafer Theory for Problems with Uncertainty/Fuzziness," *New Generation Computing* 2, Springer Verlag (1983).
 58. Kling, R., "Fuzzy Planner: Reasoning with Inexact Concepts in a Procedural Problem-Solving Language," *Journal of Cybernetics* 3, pp. 1-16 (1973).
 59. Komorowski, J., "QLOG - The Programming Environment for Prolog in LISP," in *Logic Programming*, ed. K.L. Clark and S.-A. Tarnlund, Academic Press, New York (1982). A.P.I.C. Studies in Data Processing No. 16
 60. Kowalski, Robert A., "Logic for Data Description," in *Logic and Databases*, ed. H. Gallaire, J. Minker, Plenum Press (1978).
 61. Kowalski, Robert A., *Logic for Problem Solving*, Elsevier North-Holland, New-York (1979).
 62. Kowalski, Robert A., "Logic as a Database Language," *Workshop on Logic Programming*, Long Beach, Los Angeles, Also available as Technical Report from Imperial College, London (September 1981).
 63. Lakoff, G., "Hedges: A Study in Meaning Criteria and the Logic of Fuzzy Concepts," *Journal of Philosophical Logic* 2, pp. 458-508 (1973).
 64. Lccot, Koenraad, "Probabilistic Inexact Reasoning in Prolog-Based Expert Systems," Term Paper, University of California, Los Angeles (Fall, 1983).
 65. Lee, R.C.T., "Fuzzy Logic and the Resolution Principle," *Journal of the ACM* 19, pp. 109-119 (1972).
 66. LeFaivre, R.A., "The Representation of Fuzzy Knowledge," *Journal of Cybernetics* 4, pp. 57-66 (1974).
 67. LeFaivre, R.A., "Procedural Representation in Fuzzy Problem-Solving System," *Proceedings of the National Computer Conference* (1976).
 68. Lowrance, John D., "Dependency-Graph Models of Evidential Support," COINS Report 82-26, Computer and Information Science Department, University of Massachusetts at Amherst (1982). Ph.D. Dissertation
 69. Macviar-Whelen, P.J., "Fuzzy Sets, The Concept of Height, and the Hedge "Very"," *IEEE Transactions on Systems, Man, and Cybernetics* SMC-3(6),

- Based Inference Systems," *Proceedings 1976 National Computer Conference Vol 45*, pp. 1075-1982, Also in *Readings in Artificial Intelligence*, edited by Webber and Nilsson, published by Tioga, 1981 (1976).
42. Duda, R.O., J. Gaschnig, and P.E. Hart, "Model Design in the Prospector System for Mineral Exploration," pp. 153-167 in *Expert Systems in the Microelectronic Age*, ed. D. Michie, University of Edinburgh, Scotland (1979). Also in *Readings in Artificial Intelligence*, edited by Webber and Nilsson, published by Tioga, 1981
 43. Dwiggings, Don, "Prolog as a System Design Tool," *Proceedings of the 16th Annual Hawaii International Conference on System Sciences* (1983).
 44. Dworkis, C., "Experiments in Elementary Rule-Based Reasoning," Internal Report, Operating Systems Division, Logicon, Inc (August, 1981).
 45. Fekete, G., J.O. Ecklundh, and A. Rosenfeld, "Relaxation: Evaluation and Application," *IEEE Transactions on Pattern Analysis and Machine Intelligence* 3(4), pp. 459-469 (July 1981).
 46. Fieschi, M., D. Fieschi, M. Joubert, and M. Roux, "SPHINX: An Interactive System for Medical Diagnosis," in *Fuzzy Information and Decision Processes*, ed. M. Gupta and E. Sanchez, North-Holland (1982).
 47. Freksa, Christian, "Linguistic Description of Human Judgments in Expert Systems and in the Soft Sciences," pp. 297-305 in *Approximate Reasoning in Decision Analysis*, ed. M.M. Gupta and E. Sanchez (1982).
 48. Furukawa, K., "An Intelligent Access to Relational Databases," pp. 334-349 in *Computer Science and Technologies: 1982*, ed. T. Kitagawa, Elsevier North-Holland, New-York (1982).
 49. Futo, I. and T. Gergely, "A Logical Approach to Simulation," *Proceedings of the International Conference on Model Realism*, Bad Honc, Germany (April, 1982).
 50. Gaschnig, J.G., J. Reiter, and Rene Reboh, "Development and Application of a Knowledge-Based Expert System for Uranium Resource Evaluation," SRI Technical Report, Stanford Research Institute, Menlo Park (1981).
 51. Hammond, P., "Logic Programming for Expert Systems," Master's Thesis, Imperial College, London (1980). Available as Technical Report DOC 82/4
 52. Hammond, P., "Appendix to Prolog: A Language for Implementing Expert Systems," *Machine Intelligence* 10 (1982).
 53. Hendrix, G.G., E.D. Sacerdoti, D. Sagalowics, and J. Slocum, "Developing a Natural Language Interface to Complex Data," *ACM Transactions on Database Systems* 3(2) (June 1978).
 54. Hoffman, L.J. and L.A. Neitzel, "Inexact Analysis of Risk," *Proceedings of the*

California

83. Quinlan, J.R., "INFERNO: A Cautious Approach to Uncertain Inference," *The Computer Journal* 26(3), Also available as a RAND Technical Report (August 1983).
84. Reboh, Rene, "Knowledge Engineering Techniques and Tools in the Prospector Environment," SRI Technical Report 243, Stanford Research Institute, Menlo Park (1981).
85. Reiter, R., "On Closed World Databases," pp. 55-76 in *Logic and Databases*, ed. H. Gallaire and J. Minker, Plenum Press (1978). Also in *Readings in Artificial Intelligence*, edited by Webber and Nilsson, published by Tioga, 1981.
86. Reiter, R., "A Logic for Default Reasoning," *Artificial Intelligence* 13, pp. 81-132 (October, 1980).
87. Robinson, J.A. and E.E. Sibert, "LOGLISP: An Alternative to Prolog," *Machine Intelligence*(10) (1982).
88. Roussel, P., "Prolog: Manuel de Reference et d'Utilisation," Technical Report, Groupe d'Intelligence Artificielle Marseille-Luminy (1975).
89. Schmucker, J., *Fuzzy Sets, Natural Language Computations and Risk Analysis*, Computer Science Press (1984).
90. Sergot, M., "Prospects for Representing the Law as Logic Programs," in *Logic Programming*, ed. K.L. Clark and S.-A. Tarnlund, Academic Press, New York (1982). A.P.I.C. Studies in Data Processing No. 16
91. Shafer, G., *A Mathematical Theory of Evidence*, Princeton University Press, Princeton, New Jersey (1976).
92. Shapiro, Ehud Y., *Algorithmic Program Debugging*, MIT Press (1983). Ph.D. thesis, Yale University, May 1982
93. Shapiro, Ehud Y., "Logic Programs With Uncertainties: A Tool for Implementing Rule-Based Systems," *Proceedings of the International Joint Conference on Artificial Intelligence*, Karlsruhe, Germany (1983).
94. Shortliffe, Edward H., "A Model of Inexact Reasoning in Medecine," *Mathematical Biosciences* 23, pp. 351-379 (1975).
95. Silva, Georgette and Don Dwigings, "Towards a Prolog Text Grammar," *SIGART Newsletter*(73), pp. 20-25 (October 1980).
96. Swinson, P.S.G., "Logic Programming - A Computing Tool for the Architect of the Future," *Computer-Aided Design* 14(2) (March 1982).
97. Szolovits, Peter and S.G. Pauker, "Categorical and Probabilistic Reasoning in Medical Diagnosis," *Artificial Intelligence* 11, pp. 115-144 (1978).
98. Tong, R.M. and P.P. Bonissone, "A Linguistic Approach to Decision Making

- pp. 507-511 (June, 1978).
70. McDermott, E.H., "Process Control Using Fuzzy Logic," in *Designing for Human Computer Communication*, ed. M.J. Simep and M.J. Coombs, Academic Press (1983).
 71. Markusz, Z.S., "How to Design Variants of Flats Using the Programming Language Prolog based on Mathematical Logic," *Proceedings IFIP 77*, pp. 885-889, North Holland (1977).
 72. McCloskey, M.E. and S. Glucksberg, "Natural Categories: Well Defined or Fuzzy Sets?," *Memory and Cognition* 6(4), pp. 462-472 (1978).
 73. McDermott, Drew and J. Doyle, "Nonmonotonic Logic," *Artificial Intelligence* 13(1), pp. 41-72 (November, 1980).
 74. McDermott, Drew, "The Prolog Phenomenon," *SIGART Newsletter*(72), pp. 16-20 (July 1980).
 75. Melle, W. Van, A.C. Scott, J.S. Bennet, and M. Pears, "The EMYCIN Manual," STAN-CS-81-885, Stanford University Computer Science Dept (1981).
 76. Oliveira, E., Luis Moniz Pereira, and Paul Sabatier, "ORBI, An Expert System for Environmental Resource Evaluation Through Natural Language," *Proceedings of the First International Logic Programming Conference*, Marseille, France, ADDP-GIA, Faculte des Sciences de Luminy (September 1982).
 77. Parsaye, Kamran, "Database Management, Knowledge Base Management and Expert System Development in Prolog," *Proceedings of Workshop on Logic Programming*, Algarve, Portugal (1983).
 78. Pearl, Judea, "Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach," *Proceedings of AAAI-82*, Pittsburg (August 1982).
 79. Pednault, E.P.D., S.W. Zucker, and L.V. Muresan, "On the Independence Assumption Underlying Subjective Bayesian Updating," *Artificial Intelligence* 16(3), pp. 213-222 (May, 1981).
 80. Peleg, S. and A. Rosenfeld, "A Note on the Evaluation of Probabilistic Labelings," *IEEE Transactions on Systems, Man, and Cybernetics* SCM-11(2), pp. 176-179 (February 1981).
 81. Pereira, Fernando C.N., Luis M. Pereira, and David H.D. Warren, "Prolog: The Language and its Implementation Compared with LISP," *Proceedings of the Symposium on Artificial Intelligence and Programming Languages. SIGPLAN/SIGART Notices*, Rochester, N.Y. 12(8), pp. 109-115 (August 1977).
 82. Pereira, Fernando C.N., "Logic for Natural Language Analysis," Ph.D. thesis, University of Edinburgh, U.K. (1982). Reprinted as Technical Note 275, January 1983, Artificial Intelligence Center, SRI International, Menlo Park,

APPENDICES

- with Fuzzy Sets," *IEEE Transactions on Systems, Man and Cybernetics* SMC-19(1) (1980).
99. Walker, Adrian, "Data Bases, Expert Systems, and Prolog," Technical Report RJ-3870, IBM Research Center San Jose (1982). Also in *Artificial Intelligence Applications for Business*, Edited by W. Reitman, Published by Ablex, 1984
 100. Warren, David H.D., "WARPLAN: A System for Generating Plans," DCL Memo 76, Department of Artificial Intelligence, University of Edinburgh Scotland (1974).
 101. Warren, David H.D., "Generating Conditional Plans and Program," *Proceedings of the AISB Summer Conference*, Edinburgh, Scotland (July, 1976).
 102. Warren, David H.D., "Prolog on the DECsystem-10," pp. 153-167 in *Expert Systems in the Microelectronic Age*, ed. D. Michie, University of Edinburgh, Scotland (1979).
 103. Warren, David H.D., "Logic for Compiler Writing," *Software Practice and Experience* 10(1), pp. 97-125, Also available as DAI Research Paper 44 from Department of Artificial Intelligence, University of Edinburgh (1980).
 104. Warren, David H.D., "A View of the Fifth Generation and its Impact," Research Report, SRI International, Menlo Park (1982). Also in *Artificial Intelligence Magazine*, Fall 1982
 105. Wesley, L.P., "The Use of an Evidential-Based Model for Representing Knowledge and Reasoning about Images in the VISIONS System," *Proceedings of Workshop on Computer Vision: Representation and Control*, Rindge, New Hampshire, Franklin Pierce College (1982).
 106. Zadeh, Lofti A., "Fuzzy Sets," *Information and Control* 8, pp. 338-353 (1965).
 107. Zadeh, Lofti A., "A Theory of Approximate Reasoning," *Machine Intelligence* 10 (1979).
 108. Zadeh, Lofti A., "Test-Score Semantics for Natural Language and Meaning Representation via PRUF," SRI Technical Report 247, Stanford Research Institute, Menlo Park (May 1981).
 109. Zadeh, Lofti A., "The Role of Fuzzy Logic in the Management of Uncertainty in Expert Systems," *Fuzzy Sets and Systems* 11, pp. 199-227 (November 1983).
 110. Zadeh, Lofti A., "Commonsense Knowledge Representation Based on Fuzzy Logic," *Computer* 16(10) (October 1983).
 111. Zadeh, Lofti A., "A Computational Approach to Fuzzy Quantifiers in Natural Language," *Computers and Mathematics* 9, pp. 149-184 (1983).
 112. Zaumen, W.T., "Computer Assisted Circuit Evaluation in Prolog for VLSI," *Proceedings of ACM/SIGMOD Conference on Modelling of Data*, San Jose (1983).

```

print_expert :- clear,
              nl,
              write(' ***** '),nl,
              write(' * '),nl,
              write(' *   CONSULTING THE DOMAIN EXPERT   * '),nl,
              write(' * '),nl,
              write(' ***** '),nl,
              nl.

ask_rules :-
              setof(R,rule(R),Rules),
              ask_info_rules(Rules).

ask_info_rules([]).
ask_info_rules([(X :- Y)|T]) :-
              nl,
              print_info,
              write(' What is the prior probability of '),write(X),
              write(' ? '),
              readnumber(Prob),
              assertz(prior_prob(X,Prob)),
              nl,
              write(' What is the prior probability of '),
              write(X),write(' if you know that '),
              write(Y),write(' is true'),
              write(' ? '),
              readnumber(PHE),
              assert(prior_phe(X,Y,PHE)), % P(H|E)
              nl,
              write(' What is the prior probability of '),
              write(X),write(' if you know that '),
              write(Y),write(' is false'),
              write(' ? '),
              readnumber(PHNE),
              assert(prior_phne(X,Y,PHNE)), % P(H|-E)
              ask_info_rules(T).

ask_facts :-
              setof(R,fact(R),Facts), % a fact is a piece of evidence
              ask_info_facts(Facts).

ask_info_facts([]).
ask_info_facts([X|T]) :-
              nl,
              print_info,
              write(' What is the prior probability of '),write(X),
              write(' ? '),
              readnumber(Prob),
              assertz(prior_prob(X,Prob)),
              ask_info_facts(T).

```

APPENDIX 1
PROLOG IMPLEMENTATION OF PROSPECTOR

```
% filename : prospector
% author   : Koenraad Lecot
% update   : December 5, 1983
% purpose  : Prospector probabilistic reasoning

% Prospector strategy
% -----

% Strategy
% -----

% The prior probabilities of all propositions and inference rules
% in the knowledge base are supplied by the domain expert and
% stored permanently.

% The posterior probabilities of the evidence are supplied
% by the user and are stored temporarily

% All other posterior probabilities are computed by the
% inference system and are NOT stored

% INPUT
% -----

% DEFINITION PHASE ( get all PRIOR probabilities )
% =====

% ask the DOMAIN EXPERT all necessary information needed for
% Prospector, this information consists of :
% A) For each rule H :- E
%   1) P(H|E) (prior probability) (prior_phe(H,E,PHE))
%   2) P(H|-E) (prior probability) (prior_phe(H,E,PHE))
% B) For each rule head ( hypothesis )
%   1) P(H) = uncertainty of the hypothesis (prior probability)
%      ( is stored as prior_prob(H,P) )
% C) For each piece of evidence E (non rule head)
%   1) P(E) = initial probability (prior probability)
%      ( is stored as prior_prob(E,P) )

ask_prospector_expert :-
    print_expert,
    clear_info, % get rid of all old info
    ask_rules.
```

```

% clear the database

clear_index
    abolish(post_prob,2),
    abolish(prior_prob,2),
    abolish(prior_odds,2),
    abolish(prior_phe,2),
    abolish(prior_phne,2).

% Prospector updating
% -----

% the head of a rule ( a conclusion or hypothesis )
% with possible multiple evidence
% compute the posterior probability given the user's belief
% in the various pieces of evidence

compute(Goal,Prob) :- rule_head(Goal),
    setof0(Body,clause(Goal,Body),Bodies),
    compute_likelihood(Goal,Bodies,L),
    prior_prob(Goal,Prior),
    calculate_odds(Prior,PriorOdds),
    PostOdds is L * PriorOdds,
    calculate_prob(PostOdds,Prob).

% a conjunction of predicates ( basic Zadeh fuzzy theory )

compute((A,B),Prob) :- compute(A,P1),
    compute(B,P2),
    min([P1,P2],Prob).

% a disjunction of predicates ( basic Zadeh fuzzy theory )

compute((A;B),Prob) :- compute(A,P1),
    compute(B,P2),
    max([P1,P2],Prob).

% a negation

compute(not(A),Prob) :- compute(A,P1),
    Prob is 1 - P1.

% a fact ( a basic piece of evidence )

compute(A,Prob) :- evidence(A),post_prob(A,Prob). % unit clauses
    % use the user supplied posterior probability for
    % this piece of evidence
    % ( or the default prior probability )

```

```

% CONSULTATION PHASE (PRIOR probabilities of the evidence)
% =====

% ask USER about his opinion about the various pieces of evidence
% let him provide P(Zi|Ei) (posterior probability)
% a FACT, also called a piece of EVIDENCE, is any statement
% in the knowledge base, which is not a hypothesis for
% an inference rule

ask_prospector_user :-
    print_user,
    setof(R, fact(R), Facts),
    ask_user_info_facts(Facts).

print_user :- clear,
    nl,
    write(' ***** '),nl,
    write(' * * * * * '),nl,
    write(' * CONSULTING THE USER * '),nl,
    write(' * * * * * '),nl,
    write(' ***** '),nl,
    nl.

ask_user_info_facts([]).
ask_user_info_facts([X:T]) :-
    nl,
    print_info,
    prior_prob(X,P),
    write(' The domain expert provided a probability of '),
    write(P), write(' for '), write(X),nl,
    write(' If you agree with this value'),
    write(' or you have no idea, then press CR '),nl,
    write(' otherwise enter your estimate '),
    readnumber(Prob),
    replace(Prob, post_prob(X,P)),
    ask_user_info_facts(T).

% replace old posterior probabilities of the pieces of evidence

% When the user cannot supply any new evidence, then use
% the prior probability

replace(Prob, post_prob(X,P)) :- name(Prob,[]),
    prior_prob(X,P),
    retract_check(post_prob(X,_)),
    assertz(post_prob(X,P)).

% the user supplied new evidence

replace(Prob, post_prob(X,P)) :- retract_check(post_prob(X,P)),
    assertz(post_prob(X,Prob)).

```

```

compute_li(H,E,PostPHE) :-
    compute_prior_prob(E,Prior),
    compute(E,Post),
    Prior =< Post,
    Post =< 1,
    prior_phe(H,E,PriorPHE),
    prior_prob(H,PriorPH),
    R1 is PriorPHE - PriorPH,
    R2 is 1 - Prior,
    R3 is R1/R2,
    R4 is Post - Prior,
    R5 is R3 * R4,
    PostPHE is R5 + PriorPH.

% compute the effective likelihood ratio of H|E'
% deals with multiple evidence

%
%      n
%      O(H|E') = ( * Li ) O(H)
%      i=1
% where
%
%      O(H|Ei')
%      Li = -----
%      O(H)

compute_likelihood(_,[],1).
compute_likelihood(G,[H|T],L) :-
    compute_li(G,H,P),
    calculate_odds(P,PostOdds),
    prior_prob(G,PriorProb),
    calculate_odds(PriorProb,PriorOdds),
    P1 is PostOdds / PriorOdds,
    compute_likelihood(G,T,Rest),
    L is P1 * Rest.

% Prior probabilities of boolean combinations of evidence
% a conjunction of predicates ( basic Zadeh fuzzy theory )

compute_prior_prob((A,B),Prob) :-
    compute_prior_prob(A,P1),
    compute_prior_prob(B,P2),
    min([P1,P2],Prob).

% a disjunction of predicates ( basic Zadeh fuzzy theory )

compute_prior_prob((A;B),Prob) :-
    compute_prior_prob(A,P1),
    compute_prior_prob(B,P2),
    max([P1,P2],Prob).

```



```

% Basic Bayes formulas
% -----

%      P
% O = -----
%      1 - P

% compute odds for given probability

calculate_odds(Prob,Odds) :-
    P1 is 1 - Prob ,
    Odds is Prob/P1.

% compute probability for given odds

calculate_prob(Odds,Prob) :-
    O1 is 1 + Odds,
    Prob is Odds/O1.

% compute the posterior probability of a hypothesis
% Prospector's piecewise linear function

%      { ( P(H) - P(H|~E) )
%      { P(H|~E) + ----- P(E|E')
%      {                               P(E)
%      {                               for 0 ≤ P(E|E') < P(E)
% P(H|E') = {
%      { ( P(H|E) - P(H) )
%      { P(H) + ----- ( P(E|E') - P(E) )
%      {                               1 - P(E)
%      {                               for P(E) ≤ P(E|E') ≤ 1

compute_li(H,E,PostPHE) :-
    compute_prior_prob(E,Prior),
    compute(E,Post),
    0 =< Post,
    Post < Prior,
    prior_phne(H,E,PriorPHNE),
    prior_prob(H,PriorPH),
    R1 is PriorPH - PriorPHNE,
    R2 is R1/Prior,
    R3 is R2 + Post,
    PostPHE is R3 + PriorPHNE.

```

```
retract_check(P) :- clause(P,true),!,retract(P).
retract_check((Head :- Body)) :-
    clause(Head,Body),!,retract((Head :- Body)).
retract_check(P).
```

% minimum of two elements

```
min([A,B],A) :- A<B.
min([A,B],B) :- B<A.
min([A,B],A).
```

% maximum of two elements

```
max([A,B],A) :- A>B.
max([A,B],B) :- B>A.
max([A,B],A).
```

```

% a negation
compute_prior_prob(not(A),Prob) :-
    compute_prior_prob(A,P1),
    Prob is 1 - P1.

% a fact
compute_prior_prob(A,Prob) :- prior_prob(A,Prob).

% control
% -----

start :-
    write(' Select your knowledge base '),
    readname(Name),consult2(Name),nl,
    write(' Select your inference mechanism '),
    readname(Method),
    do(Method).

do(pro prospector) :- ask_pro prospector_expert,
    ask_facts,
    ask_pro prospector_user,
    hypothesis(H),
    nl,
    write(' The posterior probability of '),
    write(H),write(' is '),
    compute(H,P),write(P).

print_pro prospector :- clear,
    nl,
    write(' ***** '),nl,
    write(' * * * * * '),nl,
    write(' * PROSPECTOR * '),nl,
    write(' * * * * * '),nl,
    write(' ***** '),nl,
    nl.

print_info :-
    write(' The probability of a statement is any '),nl,
    write(' real number between 0 and 1 where 0 means false'),nl,
    write(' and 1 means true '),nl.

% utilities
% -----

% clear screen

clear :- put(26).

% retract should not fail

```

```

% filename : consult2
% author   Koenraad Lecot
% update   December 1, 1983
% purpose  consult knowledge base

% special consult routine for loading a knowledge base
% -----

% We have to store the database references to be able to
% make a distinction between knowledge base and program

consult2(File) :-
    seeing(Input),
    see(File),
    clear_knowledge_base,
    repeat,
    read(Term),
    process(Term),
    seen,
    see(Input),!.

process(Term) :- Term = end_of_file,!.
process(?-G)  :- !,call(G),!,fail.
process(Clause) :- assertz(Clause,R),
                  assertz(ref(R)), % store the db reference too
                  fail.

% UTILITIES
% -----

% find the head of a rule

head(( X :- Y),X).

% find the body of a rule

body(( X :- Y),Y).

% distinction between facts and rules

fact(F) :- ref(R),clause(F,true,R).
fact(F) :- rule_body(B),and_member(F,B),not(rule_head(F)).

evidence(E) :-
    fact(E), % definition of a piece of evidence
    not(rule_head(E)).

rule(( X :- Y)) :- ref(R),clause(X,Y,R),Y == true.

rule_head(X) :- ref(R),clause(X,Y,R),Y == true.

```

```

% filename : read
% author   : Gerard Lecot
% update   : November 21, 1983
% purpose  : read real numbers

% READ routines
% -----

readnumber(N) :-
    nl,prompt(X,' give a number > '),
    readstring(List),
    reverse(List,[H|T]),
    reverse(T,List2), % get rid of CR at the end
    name(N,List2).

readname(N) :-
    nl,prompt(X,' input please > '),
    readstring(List),
    reverse(List,[H|T]),
    reverse(T,List2), % get rid of CR at the end
    name(N,List2).

readstring([K|U]) :- get0(K1),readrest(K1,U).

readrest(10,[]) :-!. /* newline */
readrest(12,[]) :-!. /* newline */
readrest(13,[]) :-!. /* newline */
readrest(14,[]) :-!. /* newline */

readrest(K,[K|U]) :- K<33,!,get(K1),readrest(K1,U).

readrest(K1,[K2|U]) :- get0(K2),readrest(K2,U).

% utilities
% -----

reverse(L1,L2) :- revzap(L1,[],L2).
revzap([X|L],L2,L3) :- revzap(L,[X|L2],L3).
revzap([],L,L).

```

```
retract_list(T1).

% Search for a set of things, like setof, but can return nil

setof0(Expr, SearchExpr, Answer) :-
    setof(Expr, SearchExpr, Answer), !.
setof0(Expr, SearchExpr, []).
```

```

rule_body(X) :- ref(R), clause(X,Y,R), Y == true.

% and_member : check if a predicate is a member of a conjunction

and_member(P,(P,Q)).
and_member(P,(P1,Q)) :- !, and_member(P,Q).
and_member(P,P).

% transform a conjunction of predicates into a list of predicates

and_to_list((X,Y),[X|Z]) :-!, and_to_list(Y,Z).
and_to_list(true,[]) :- !.
and_to_list(X,[X]) :- !.

% ancestor : find an ancestor of a goal in an AND/OR tree
% A is ancestor of G

ancestor(G,A) :- rule(R), body(R,RB), and_member(G,RB), head(R,A).

% parent : find the parent goal of a goal
% P is parent of G

parent(G,P) :- ancestor(G,P).

% child : find a child of a node in the AND/OR tree
% C is child of P if P is parent of C

child(P,C) :- parent(C,P).

% hypothesis = a rule head without an ancestor

hypothesis(H) :- rule_head(H), !, setof(X, ancestor(H,X), []).

% utilities
% -----

% clear the knowledge base

clear_knowledge_base :-
    setof(F, fact(F), Facts),
    retract_list(Facts),
    setof(R, rule(R), Rules),
    retract_list(Rules),
    abolish(ref, 1),
    abolish(prior_odds, 2).

% retract all clauses in a flat list

retract_list([]).
retract_list([H1|T1]) :-
    retract_check(H1),

```

APPENDIX 2
EXECUTION TRACE OF PROSPECTOR IN PROLOG

Script started on Wed Dec 7 21:49:15 1983
cprolog
C-Prolog version 1.4a, Silogic release A
Copyright University of Edinburgh and Silogic, Inc. 1983
! ?- [begin].
prospector consulted 6820 bytes 2.8 sec.
read consulted 1068 bytes 0.483334 sec.
matrix consulted 4112 bytes 1.45 sec.
consult2 consulted 2440 bytes 0.950001 sec.
No leashing.
Debug mode switched off.

yes
begin consulted 14440 bytes 6.6 sec.

yes
! ?- start.
Select your knowledge base
input please > real

Select your inference mechanism
input please > prospector

*
* CONSULTING THE DOMAIN EXPERT *
*

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true

What is the prior probability of file ?
give a number > 0.23

What is the prior probability of file if you know
that cvr is true ?
give a number > 0.52

What is the prior probability of file if you know
that cvr is false ?
give a number > 0.25

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true


```

% filename : real

% Prospector example inference net
% Example from [Ginsburg81]

% gir = granitic intrusives in region
% fre = favorable regional environment
% fle = favorable level of erosion
% oftsys = preintrusive throughgoing fault system
% cvr = coeval volcanic rocks
% hype = hypabyssal regional environment
% stir = suggestive texture of igneous rock
% smir = suggestive morphology of igneous rock
% fmgf = fine-to-medium grain size
% pt = porphyritic texture
% rcs = stocks
% rcad = dikes
% rcib = intrusive breccias
% rcvp = volcanic plugs

gir :- fre.
fre :- fle.
fre :- oftsys.
fle :- cvr.
fle :- hype.
hype :- stir.
stir :- fmgf.
stir :- fmgf,pt.
fmgf :- pt.
hype :- smir.

smir :- rcib.
smir :- rcs.
smir :- rcad.
smir :- rcvp.
smir :- rcib;rcs;rcad;rcvp.

```

give a number > 0.54

The probability of a statement is any real number between 0 and 1 where 0 means false and 1 means true

What is the prior probability of gir ?

give a number > 0.63

What is the prior probability of gir if you know that fre is true ?

give a number > 0.62

What is the prior probability of gir if you know that fre is false ?

give a number > 0.64

The probability of a statement is any real number between 0 and 1 where 0 means false and 1 means true

What is the prior probability of hype ?

give a number > 0.63

What is the prior probability of hype if you know that smir is true ?

give a number > 0.72

What is the prior probability of hype if you know that smir is false ?

give a number > 0.72

The probability of a statement is any real number between 0 and 1 where 0 means false and 1 means true

What is the prior probability of hype ?

give a number > 0.74

What is the prior probability of hype if you know that stir is true ?

give a number > 0.42

What is the prior probability of hype if you know that stir is false ?

give a number > 0.35

The probability of a statement is any real number between 0 and 1 where 0 means false and 1 means true

What is the prior probability of smir ?

give a number > 0.97

What is the prior probability of smir if you know

What is the prior probability of fle ?
give a number > 0.12

What is the prior probability of fle if you know
that hnye is true ?
give a number > 0.23

What is the prior probability of fle if you know
that hnye is false ?
give a number > 0.75

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true
What is the prior probability of fngs ?
give a number > 0.01

What is the prior probability of fngs if you know
that pt is true ?
give a number > 0.91

What is the prior probability of fngs if you know
that pt is false ?
give a number > 0.52

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true
What is the prior probability of fre ?
give a number > 0.63

What is the prior probability of fre if you know
that fle is true ?
give a number > 0.47

What is the prior probability of fre if you know
that fle is false ?
give a number > 0.52

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true
What is the prior probability of fre ?
give a number > 0.19

What is the prior probability of fre if you know
that ofsys is true ?
give a number > 0.23

What is the prior probability of fre if you know
that ofsys is false ?

and 1 means true

What is the prior probability of smir ?

give a number > 0.23

What is the prior probability of smir if you know
that rcib;rcs;rcad;rcvp is true ?

give a number > 0.98

What is the prior probability of smir if you know
that rcib;rcs;rcad;rcvp is false ?

give a number > 0.34

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true

What is the prior probability of stir ?

give a number > 0.98

What is the prior probability of stir if you know
that fmgs is true ?

give a number > 0.23

What is the prior probability of stir if you know
that fmgs is false ?

give a number > 0.32

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true

What is the prior probability of stir ?

give a number > 0.12

What is the prior probability of stir if you know
that fmgs,pt is true ?

give a number > 0.21

What is the prior probability of stir if you know
that fmgs,pt is false ?

give a number > 0.45

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true

What is the prior probability of cvr ?

give a number > 0.65

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true

What is the prior probability of otfsys ?

give a number > 0.12

that road is true ?
give a number > 0.23

What is the prior probability of smir if you know
that road is false ?
give a number > 0.97

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true
What is the prior probability of smir ?
give a number > 0.12

What is the prior probability of smir if you know
that rcib is true ?
give a number > 0.34

What is the prior probability of smir if you know
that rcib is false ?
give a number > 0.54

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true
What is the prior probability of smir ?
give a number > 0.63

What is the prior probability of smir if you know
that rcs is true ?
give a number > 0.23

What is the prior probability of smir if you know
that rcs is false ?
give a number > 0.87

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true
What is the prior probability of smir ?
give a number > 0.65

What is the prior probability of smir if you know
that rcvp is true ?
give a number > 0.97

What is the prior probability of smir if you know
that rcvp is false ?
give a number > 0.65

The probability of a statement is any
real number between 0 and 1 where 0 means false

real number between 0 and 1 where 0 means false
and 1 means true
The domain expert provided a probability of 0.12 for outsys
If you agree with this value or you have no idea, then press CR
otherwise enter your estimate
give a number > 0.75

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true
The domain expert provided a probability of 0.86 for pt
If you agree with this value or you have no idea, then press CR
otherwise enter your estimate
give a number > 0.32

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true
The domain expert provided a probability of 0.53 for road
If you agree with this value or you have no idea, then press CR
otherwise enter your estimate
give a number > 0.23

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true
The domain expert provided a probability of 0.87 for rcib
If you agree with this value or you have no idea, then press CR
otherwise enter your estimate
give a number > 0.20

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true
The domain expert provided a probability of 0.43 for rcs
If you agree with this value or you have no idea, then press CR
otherwise enter your estimate
give a number >

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true
The domain expert provided a probability of 0.63 for rcvp
If you agree with this value or you have no idea, then press CR
otherwise enter your estimate
give a number > 0.1

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true
The domain expert provided a probability of 0.62

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true
What is the prior probability of pt ?
give a number > 0.86

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true
What is the prior probability of read ?
give a number > 0.53

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true
What is the prior probability of rcib ?
give a number > 0.87

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true
What is the prior probability of rcs ?
give a number > 0.43

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true
What is the prior probability of rcvp ?
give a number > 0.63

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true
What is the prior probability of rcib;rcs;rcad;rcvp ?
give a number > 0.62

```
*****  
*                                     *  
*           CONSULTING THE USER       *  
*                                     *  
*****
```

The probability of a statement is any
real number between 0 and 1 where 0 means false
and 1 means true
The domain expert provided a probability of 0.65 for cvr
If you agree with this value or you have no idea, then press CR
otherwise enter your estimate
give a number >

The probability of a statement is any

APPENDIX 3
PROLOG IMPLEMENTATION OF MYCIN

```
% Filename: mycin.pl
% Author   : Peter Hammond / Koenraad Lecot
% Date    : February 10, 1984
% Subject  : Prolog Implementation of Mycin

% *****
% *
% *      Mycin in Prolog      *
% *
% *****

% Original program, written by Peter Hammond, 1980
% Debugged, converted from Waterloo Prolog to CProlog
% by Koenraad Lecot, 1984

% Extensions include :

% 1. Better I/O interface, and error recovery.
% 2. Modified the explanation facility to make it more general.
% 3. Modified the dataclass datatype to make the whole program
%    more independent from its application area, i.e. a first step
%    toward EMycin
% 4. Using more meaningful names for predicates and variables
% 5. Document the program

% The data structures included
% in the Prolog version of Mycin are a
% knowledge base, i.e. rules,
% a context tree to store background data
% and important facts obtained during the
% consultation phase and finally, a
% separate record of the possible identities
% of each suspected organism.

% The context tree is defined by a datatype
% that reflects the hierarchical
% structure
% dataclass(class-number,class-name,class-details)

% A record of possible identities of each organism is kept by
% know(<organism>,<identity>,<certainty-factor>)

% rule : embodies the actual knowledge base

% The consultation system
```



```
for rcib;rcs;rcad;rcvp
If you agree with this value or you have no idea, then press CR
otherwise enter your estimate
give a number >
```

```
The posterior probability of gir is 0.63521
```

```
| ?- halt.
```

```
[ Prolog execution halted ]
```

```
# ^D
```

```
script done on Wed Dec 7 21:53:14 1983
```

```

% gen_no : generates a unique number for each new entity

gen_no(Class,NewNo) :-
    retract(number(Class,OldNo)),
    sum(1,OldNo,NewNo),
    assert(number(Class,NewNo)).

genheader(Entity) :-
    write(' '),write(Entity),
    nl.

% get_details : asks the user to supply information of a
% background nature for a particular entity and
% can also cause a message to be printed
% announcing the name of the first entity in the
% next class in the hierarchy

get_details(Class,Entity,Descr) :-
    dataclass(Class,Name,Details),
    get_list(Class,Details,Descr),
    generate_messages(Class,Entity).

% We note again that the datatype dataclass embodies the
% hierarchical structure of the context tree

get_list(Class,[],[]).
get_list(Class,[Item|OtherItems],[Answer|OtherAnswer]) :-
    get_answer(Class,Item,Answer),
    get_list(Class,OtherItems,OtherAnswer).

% current : returns the current entity of a particular class

current(Class,Entity) :-
    number(Class,No),
    dataclass(Class,ClassName,Details),
    concat(ClassName,No,Entity).

% number : stores the number of a current entity
% for a particular class
% This declaration serves as initial data for the gen_no predicate
% described earlier.

number(0,0).
number(1,0).
number(2,0).
number(3,0).
number(question,0).
number(rule,0).

% get_answer : asks the user for data and reads the value
% of an item of background data.

```

```

% =====

% 1. Top Level ( initiation by start_session )

start_session :- therapy_required,consultation.
start_session :- close_down.

% therapy_required will ask the user if he has obtained positive
% cultures, so that the system will know if the consultation
% session should continue or not.

therapy_required :-
    print_header,
    get_answer(0,initiator,Answer),!,
    Answer = yes.

% The call get_data(0,PatientData) starts the consultation fase

consultation :- get_data(0,PatientData),
    nl,
    write(' The patient data obtained during this session are:'),
    nl,
    write_out(PatientData).

% get_data : get patient data, to build a context tree

get_data(Class1,[Class1DataItem|OtherClass1Data]) :-
    lt(Class1,4),
    gen_new(Class1,Entity),
    get_details(Class1,Entity,Descr),
    sum(Class1,1,Class2),
    get_data(Class2,Class2Data),
    Class1DataItem =.. [Entity,Descr,Class2Data],
    check_for_more(Class1,OtherClass1Data).

% the end of the recursion in the data gathering process

get_data(4,Hypothesis) :-
    current(3,Entity),
    get_hypothesis(Entity,genus,Hypothesis).
% use the given knowledge base

% gen_new : generates a new entity of the required class and
% also causes the printing of a headline announcing the new entity

gen_new(Class,Entity) :-
    gen_no(Class,NewNo),
    dataclass(Class,ClassName,Details),
    concat(ClassName,NewNo,Entity),
    genheader(Entity).

```

```

% The rest of the predicates in this section deal explicitly
% with the predefined knowledge base. The predicate same
% is used in the rules that are described at the end.
% Confidence factors are entered or defined on a scale of
% [-1000,+1000].
% same : causes evidence to be gathered which bears on a
% value of a clinical parameter and succeeds if the CF
% supporting this value is greater than 200, a threshold
% determined by the medical expert.

same(Entity,Attribute,RequiredValues,MaxCF) :-
    get(Entity,Attribute,RequiredValues, MaxCF),!,
    gt(MaxCF,200).

% note the use of "cut" in the above definition as we do not want
% the process of evidence collection to be repeated if the
% CF of the result is less than 200

% same is defined in terms of get, a function that will compute
% the CF of the hypothesis, supported by Entity and Attribute

% get: collects the hypotheses relevant to determining
% the value of clinical
% parameter and finds the largest of the confidence
% factors supporting the possible values

get(Entity,Attribute,RequiredValues,MaxCF) :-
    get_hypothesis(Entity,Attribute,Hypotheses),
    intersect(Hypotheses,RequiredValues,Intersection),
    max_hypothesis(Intersection,MaxCF).

% get_hypothesis is the most important procedure.
% We have three possible methods of determining an
% attribute value:

% 1. First, we can test if it already known in the current context

get_hypothesis(Entity,Attribute,[Hypothesis]) :-
    know(Entity,Attribute,[Hypothesis]).

% 2. Next, we can try to deduce it from the knowledge base

get_hypothesis(Entity,Attribute,[Hypothesis]) :-
    deduce(Entity,Attribute,[Hypothesis]).

% 3. Finally, we may ask the user to supply it. We note that the
% user always has the option to enter "unknow"

get_hypothesis(Entity,Attribute,[Hypothesis]) :-
    ask_for(Entity,Attribute,[Hypothesis]).

```

```
% Each question is preceded by a new question
% number
```

```
get_answer(Class,Item,Answer) :-
    gen_no(question,Q),
    writeno(Class,Q),
    question(Item),
    readstr(Answer),nl.
```

```
writeno(Class,Q) :-
    write(' '),
    write('('),
    write(Q),
    write(')').
```

```
% check_for_more : asks the user if there is another entity of a
% particular class to be considered
```

```
check_for_more(Class,OtherData) :-
    dataclass(Class,ClassName,Details),
    write(' Is there another '),
    write(ClassName),write(' ? '),nl,
    write(' CR means no to me '),write('? '),
    readstr(Answer2),
    ( Answer2 = unknown -> Answer = no ; Answer = Answer2 ),
    nl,
    consider(Answer,Class,OtherData).
```

```
consider(no,Class,[]).
consider(yes,Class,OtherData) :-
    get_data(Class,OtherData).
```

```
% The call get_data, in the previous procedure will continue
% the consultation process at any level in the context tree
% concat : concatenate two strings, using a hyphen
```

```
concat(M,N,L) :-
    name(M,List1),
    name(N,List2),
    name('-',Hyphen),
    append(List1,Hyphen,List3),
    append(List3,List2,List4),
    name(L,List4).
```

```
% check if string A is a substring of B
```

```
substring(A,B) :-
    name(A,List1),
    name(B,List2),
    append(List1,_,List2).
```

```

diff(1000,CF1,CF4),
product(CF4,CF2,CF5),
sum(CF3,CF1,CF3).

intersect([v(Value,CF)|H],[Value|Rest],[v(Value,CF)|H1]) :-
    intersect(H,Rest,H1).
intersect([v(Value,CF)|H],[M|Rest],H1) :-
    ne(Value,M),
    intersect(H,[M],X),
    intersect([v(Value,CF)|H],Rest,Y),
    union(X,Y,H1).

intersect([],Anything,[]).
intersect(Anything,[],[]).

union([],Y,Y).
union(X,[],X).
union([R],Y,[R|Y]).

% max_hypothesis computes the maximum of the confidence factors.

max_hypothesis([],0).
max_hypothesis([v(V,C)|H],MaxCF) :-
    max_hypothesis(H,C1),
    largest(C1,C,MaxCF).

largest(CF1,CF2,CF1) :-
    gt(CF1,CF2).
largest(CF1,CF2,CF2) :-
    gt(CF2,CF1).

% ask_for: causes the user to be asked for
% the value of a clinical parameter
% The answer is read and a check is made to see
% if the answer is legal

ask_for(Entity,Attribute,[v(ActualValue,CF)]) :-
    question(Entity,Attribute),
    parameter(Attribute,ExpectedValues),
    nl,
    writeln(' Please enter one of the following values: '),
    write(' '),
    write_list(ExpectedValues),
    write(' Default, when you press CR, is the value unknown '),nl,
    read_answer(Answer1,CF1),
    check_for_query(Answer1,CF1,A,C),
    check_answer(Attribute,A,C,ActualValue,CF),
    assert(!know(Entity,Attribute,[v(ActualValue,CF)])).

check_answer(Attribute,A1,C1,A1,C1) :-
    parameter(Attribute,ExpectedValues),

```

```

% We should also note that the real Mycin makes use of metarules
% in its definition of get_hypothesis. We did not include this
% powerful feature in our implementation, due to lack of time
% and the sheer size of the knowledge base

% deduce : collects all the evidence for the value of a parameter,
% merges evidence for the same value into one hypothesis
% and stores the information obtained
% by calling rule-check in a setof, deduce will inspect the whole
% knowledge base

deduce(Entity,Attribute,Hypotheses) :-
    setof(v(Value,CF),
        rule_check(Entity,Attribute,Value,CF),Hypothesis1),
        merge(Hypothesis1,Hypotheses),
        assert(known(Entity,Attribute,Hypotheses)).

% rule_check : investigates a rule for applicability
% and calculates the CF of the deduction when the rule
% succeeds

rule_check(Entity,Attribute,Value,CF) :-
    is_rule(RuleNo),
    asserta(currentrule(RuleNo)), % for explanation purposes
    rule(RuleNo,Entity,Attribute,Value,C,Tally),
    product(C,Tally,CF).

% The above call "rule(RuleNo,Entity,Attribute,Value,C,Tally)"
% will backtrack until an applicable rule is found

product(A,B,C) :- prod(A,B,D),quot(D,1000,C).

% merge : causes evidence for the same value of an attribute to be
% combined together to form a single hypothesis

merge([],[]).
merge([v(unknown,1000)],[]).
merge([H|Rest],[H1|R1]) :-
    compare_lists(H,Rest,H1,S1),
    merge(S1,R1).

compare_lists(R,[],R,[]).
compare_lists(v(Value,CF1),[v(Value,CF2)|U],R,W) :-
    new_cf(CF1,CF2,CF3),
    compare_lists(v(Value,CF3),U,R,W).
compare_lists(v(Value1,CF1),
    [v(Value2,CF2)|U],R,[v(Value2,CF2)|W]) :-
    eq(Value1,Value2),
    compare_lists(v(Value1,CF1),U,R,W).

new_cf(CF1,CF2,CF3) :-

```

```

% stack mechanism, implemented by current_rule,
% to simulate the same
% behavior.

% check_for_query is called after the user is asked
% to give a parameter
% value. If either WAF or RULE is input
% then the EXPLANATION SYSTEM is used.

check_for_query(Answer,CF,A,C) :-
    member(Answer,[why,rule]),
    answer_query(Answer),
    get_nearest(Rule),
    report(Answer,Rule),
    check_again_for_query(A,C).

check_for_query(Answer,CF,A,C) :-
    A = Answer , C = CF. % i.e. no special action is taken

answer_query(why) :-
    write('Asking all this to determine the genus of the organism '),
    nl.
answer_query(rule) :-
    write('Current rule is ').

check_again_for_query(A,C) :-
    read_answer(A1,C1),!,
    test(A1,C1,A,C).

test(why,1000,A,C) :- fail.
test(A,C,A,C) :- eq(why,A).

get_nearest(rule(RuleNo,E,A,R,C,T)):-
    currentrule(RuleNo),
    rule(RuleNo,E,A,R,C,T).

report(why,Rule) :- explain(Rule).
report(rule,Rule) :- clause(Rule,Body),
    translate((Rule:-Body)).

explain(Head) :- clause(Head,Body),
    divide(Body,Knowprems,Unknownprems),
    write_known(Knowprems),
    translate((Head :- Unknownprems)).

divide((A,B),(A,Otherknown),Unknown) :-
    known(A),
    divide(B,Otherknown,Unknown).
divide((A,B),[],(A,B)) :-
    known(A).
divide(A,[],A) :- !,known(A).

```



```

member(A1,ExpectedValues).

check_answer(Answer,Attribute,A1,C1,A,C) :-
    parameter(Attribute,ExpectedValues),
    write('I am sorry, but we will have to this part on the '),
    write(Attribute),
    write(' over again'),nl,
    write(' You will have to reenter the value and CF '),nl,
    write(' Please enter one of the following values: '),
    write(' '),
    write_list(ExpectedValues),
    read_answer(A2,C2),
    check_answer(Attribute,A2,C2,A,C).

question(Entity,Attribute) :-
    write(' Enter the '),
    write(Attribute),
    write(' of '),
    write(Entity).

% read_answer :
% is used to read to user's reply to a request for the
% value of a clinical parameter. The value and its CF
% are both read ( the default CF is 1000 )

read_answer(Answer,CF) :-
    readstr(Answer),nl,
    write('Enter the CF on scale [-1000,+1000], default 1000 (CR)'),
    nl,readstr(Something),
    ( Something = unknown -> CF = 1000 ; CF = Something ).

% The Explanation System
% =====

% The Mycin queries why and rule are easily implementable in
% Waterloo Prolog which has a system predicate "ancestor".
% This predicate can be
% used to examine the ancestors of the literals which invoked the
% predicate. When ancestor is used with one argument, its argument
% is unified with the most recent ancestor
% for which this is possible.
% If the predicate succeeds and
% subsequently backtracking returns to
% this point in the proof,
% then the argument is unified with the next
% most recent ancestor, and so forth.
% This latter feature is most useful
% in the repeated use of "why".
% As CProlog does not have this ancestor predicate,
% we used a global

```

```

        max_hypothesis(I,MaxCF),
        gt(MaxCF,200).
known(!).

rule_no(M) :-
    write(' ( We used rule '),
    write(M),
    writeln(' )').

write_value([M]) :- write(M).

give_evidence(C) :- ge(C,800),
    write(' strongly suggestive '),
    write('( '),write(C),write(')').

give_evidence(C) :- ge(C,400),
    write(' suggestive '),
    write('( '),write(C),write(')').

give_evidence(C) :- write(' weakly suggestive '),
    write('( '),write(C),write(')').

% Output Routines
% =====

% close_down : informs the user that a consultation is unnecessary

close_down :- writeln(' Patient does not require therapy').

% dataclass : enables the user to declare the classes of data,
% their names and the background details required.
% There are no 'class-details' for 'organism' because at this
% class level, we begin to use the knowledge base to obtain more
% specific information from the user.

dataclass(0,patient,[name,sex,age]).
dataclass(1,infection,[infection_type,infection_date]).
dataclass(2,culture,[culture_site,culture_date]).
dataclass(3,organism,[]).

generate_messages(0,Entity).
generate_messages(1,Entity) :-
    write(' The most recent culture associated with '),
    write(Entity),
    writeln(' will be referred to as : ').
generate_messages(2,Entity) :-
    write(' The first significant organism from '),
    write(Entity),
    writeln(' will be referred to as : ').
generate_messages(3,Entity).

question(name) :-
    writeln(' patient name :').

```

```

write_known([]).
write_known((A,B)) :-
    writeln(' It is know that :'),
    write_premise((A,B)),
    writeln(' therefore ').
write_premise([]).
write_premise((A,B)) :-
    translate_predicate(A),
    write_premise(B).
write_premise(A) :- translate_predicate(A).

% translate : gives a simple English translation of a rule

translate((Head :- Body)) :-
    Body = (I,min(_,_)), % nothing to explain at this point
    writeln(' we conclude : '),
    translate_predicate(Head).
translate((Head :- Body )) :-
    writeln(' if : '),
    write_premise(Body),
    writeln(' then : '),
    translate_predicate(Head).

translate_predicate(I). % no need to translate this
translate_predicate(min(M,N)). % no need to translate this
translate_predicate(same(E,A,R,C)) :-
    current(3,Entity),
    write(' the '),
    write(A),write(' of '),
    write(Entity),
    write(' is '),
    write_value(R),
    nl.

translate_predicate(rule(N,E,A,V,C,T)) :-
    current(3,Entity),
    write(' There is'),
    give_evidence(C),
    write(' evidence that the '),
    nl,
    write(' '),
    write(A),
    write(' of '),
    write(Entity),
    write(' is '),
    writeln(V),
    rule_no(N),
    nl.

known(same(E,A,R,C)) :-
    know(E,A,Hypothesis),
    intersect(Hypothesis,R,I),

```

```

write(' '),write_class_data_item(X),nl,write_out(Y).

write_class_data_item([]).
write_class_data_item(ClassDataItem) :-
    ClassDataItem =.. [Entity,Details,[Class2Data]],
    write_entity(Entity),
    write_details(Entity,Details),
    write_class_data_item(Class2Data).
write_class_data_item(v(Value,CF)):-
    nl,nl,
    write(' DIAGNOSIS OF GENUS '),nl,
    write(' value = '),write(Value),nl,
    write(' '),write(' CF = '),write(CF).

write_entity(Entity) :-
    nl,
    write(' ENTITY : '),
    write(Entity).

write_details(_,[]) :-
    nl.
write_details(Entity,Details) :-
    substring(patient,Entity),
    nl,
    write(' DETAILS : '),
    Details = [Name,Sex,Age],
    write('name of patient : '),write(Name),nl,write(' sex : '),
    write(Sex),write(' age : '),write(Age).
write_details(Entity,Details) :-
    substring(infection,Entity),
    nl,
    write(' DETAILS : '),
    Details = [Name,Date],
    write('name infection : '),write(Name),nl,
    write(' date first appearance : '),write(Date).
write_details(Entity,Details) :-
    substring(culture,Entity),
    nl,
    write(' DETAILS : '),
    Details = [Name,Date],
    write('name culture : '),write(Name),nl,
    write(' date obtained : '),write(Date).

% arithmetic operations

lt(X,Y) :- X < Y.
le(X,Y) :- X =< Y.
gt(X,Y) :- X > Y.
sum(X,Y,Z) :- Z is X + Y.
diff(X,Y,Z) :- Z is X - Y.

```

```

question(sex) :- writeln(' sex :').
question(age) :- writeln(' age :').
question(initiator):-
    writeln(' have you been able to obtain'),
    writeln(' cultures from a site at which the patient'),
    writeln(' has an infection ').
question(infection_type) :- writeln(' what is the infection').
question(infection_date) :-
    writeln(' please give the date when this'),
    writeln(' infection first appeared ( da/no/yr )').
question(culture_site):-
    writeln(' from what site was the specimen for this'),
    writeln(' culture taken ? ').
question(culture_date) :-
    writeln(' please give the date when this'),
    writeln(' culture was obtained ( da/no/yr )').

parameter(genus,[unknown,strept,neisseria,bact,staph,corym]).
parameter(gramstain,[unknown,pos,neg]).
parameter(morphology,[unknown,rod,coccus]).
parameter(conformation,[unknown,singles,longchains,shortchains]).
parameter(aerobicity,[unknown,anaerobic,facul]).

hypothesis(pete,ill,[]).

% Utilities
% =====

% initial header message

print_header :-
    clear_screen,nl,
    write(' ***** '),nl,
    write(' * * '),nl,
    write(' * Welcome to Mycin * '),nl,
    write(' * * '),nl,
    write(' ***** '),nl.

clear_screen :- put(12). % does not work on Wyse-50

writeln(X) :- write(X),nl.

write_list([]) :- nl.
write_list([X|Y]) :- write(X),write(' , '),write_list(Y).

% special output procedure for patient data

write_out([]) :- nl.
write_out([X|Y]) :-

```

```
same(Entity, conformation, [longchains], CF3), 1,  
min([CF1, CF2, CF3], Tally).  
  
rule(413, Entity, genus, strept, 800, Tally) :-  
same(Entity, gramstain, [pos], CF1), 1,  
same(Entity, morphology, [coccus], CF2), 1,  
same(Entity, conformation, [shortchains], CF3), 1,  
min([CF1, CF2, CF3], Tally).  
  
is_rule(9).  
is_rule(35).  
is_rule(306).  
is_rule(412).  
is_rule(413).
```

```

prod(X,Y,Z) :- Z is X * Y.
quot(X,Y,Z) :- Z is X / Y.

min([X],X).
min([X|Y],Z) :- min(Y,U),least(X,U,Z).
least(X,Y,X) :- le(X,Y).
least(X,Y,Y) :- le(Y,X).

% comparing terms

eq(X,Y) :- X = Y.
ne(X,Y) :- not X = Y.

% *****
% *
% * RULES IN THE Mycin ( PROLOG ) KNOWLEDGE BASE *
% *
% *****

rule(9,Entity,genus,neisseria,800,Tally) :-
    same(Entity,gramstain,[neg],CF1),!,
    same(Entity,morphology,[coccus],CF2),!,
    min([CF1,CF2],Tally).

% In the above rule, the call same(Entity,gramstain,[neg],CF1)
% causes all the evidence bearing on the gramstain of 'entity'
% to be collected. If the certainty factor of the hypothesis which
% suggests the gramstain is negative is larger than 200, then CF1
% takes this value. Otherwise same fails and the cut prevents
% the collection of evidence being repeated. If each clause in the
% premise succeeds, then Tally takes the value of the weakest of
% the certainty factors.
% We note here that the definition of same also contains a cut
% to avoid the repetition of unnecessary evidence collection.

rule(35,Entity,genus,bact,600,Tally) :-
    same(Entity,gramstain,[neg],CF1),!,
    same(Entity,morphology,[rod],CF2),!,
    same(Entity,aerobicity,[anaerobic],CF3),!,
    min([CF1,CF2,CF3],Tally).

rule(306,Entity,genus,staph,700,Tally) :-
    same(Entity,gramstain,[pos],CF1),!,
    same(Entity,morphology,[coccus],CF2),!,
    same(Entity,conformation,[singles],CF3),!,
    min([CF1,CF2,CF3],Tally).

rule(412,Entity,genus,strept,950,Tally) :-
    same(Entity,gramstain,[pos],CF1),!,
    same(Entity,morphology,[coccus],CF2),!,

```

(7) from what site was the specimen for this culture taken ?

? armpit

(8) please give the date when this culture was obtained (da/mo/yr)

? 15september1993

The first significant organism from culture-1 will be referred to as :

organism-1

Enter the granstain of organism-1

Please enter one of the following values:

unknown , pos , neg ,

Default, when you press CR, is the value unknown

? pos

Enter the CF on a scale of [-1000,+1000], default is 1000 (CR)

? 850

Enter the morphology of organism-1

Please enter one of the following values:

unknown , rod , coccus ,

Default, when you press CR, is the value unknown

? coccus

Enter the CF on a scale of [-1000,+1000], default is 1000 (CR)

? 700

Enter the conformation of organism-1

Please enter one of the following values:

unknown , singles , longchains , shortchains ,

Default, when you press CR, is the value unknown

? singles

Enter the CF on a scale of [-1000,+1000], default is 1000 (CR)

?

Is there another organism ?

CR means no to me ?

Is there another culture ?

CR means no to me ?

Is there another infection ?

CR means no to me ?

Is there another patient ?

CR means no to me ?

The patient data obtained during this session are:

ENTITY : patient-1

APPENDIX 4
EXECUTION TRACE OF MYCIN IN PROLOG

Script started on Sat Mar 17 20:10:44 1984
Warning: no access to tty; thus no job control in this shell...
% cprolog -u
| ?- [start].
mycin.pl consulted 17000 bytes 7.799999 sec.
/p/i/koen/prologlib/io consulted 10460 bytes 4.500004 sec.
/p/i/koen/prologlib/lists consulted 8136 bytes 2.950006 sec.
/p/i/koen/prologlib/general consulted 1492 bytes 0.555578 sec.
No leashing.
util consulted 20038 bytes 8.255553 sec.
start consulted 37132 bytes 16.203322 sec.

yes
| ?- start_session.

```
*****  
*                               *  
*   Welcome to MYCIN           *  
*                               *  
*****
```

(1) have you been able to obtain
cultures from a site at which the patient
has an infection

? yes

patient-1
(2) patient name :
? koenraad

(3) sex :
? male

(4) age :
? 27

infection-1
(5) what is the infection
? bacteremia

(6) please give the date when this
infection first appeared (da/mo/yr)
? 12june1983

The most recent culture associated with infection-1
will be referred to as :
culture-1

culture taken ?
? armpit

(8) please give the date when this
culture was obtained (da/no/yr)
? 12march1981

The first significant organism from culture-2
will be referred to as :

organism-2

Enter the gramstain of organism-2

Please enter one of the following values:

unknown , pos , neg ,

Default, when you press CR, is the value unknown

? neg

Enter the CF on a scale of [-1000,+1000], default is 1000 (CR)

? 900

Enter the morphology of organism-2

Please enter one of the following values:

unknown , rod , coccus ,

Default, when you press CR, is the value unknown

? why

Enter the CF on a scale of [-1000,+1000], default is 1000 (CR)

?

We are asking all this to determine the genus of the organism

Enter the morphology of organism-2

Please enter one of the following values:

unknown , rod , coccus ,

Default, when you press CR, is the value unknown

? coccus

Enter the CF on a scale of [-1000,+1000], default is 1000 (CR)

? 990

It is know that :

the gramstain of organism-2 is neg

the morphology of organism-2 is coccus

therefore

we conclude :

There is weakly suggestive (800) evidence that the

genus of organism-2 is neisseria

(We used rule 9)

?stop

yes

! ?-

script done on Sun Mar 18 10:57:43 1984

DETAILS : name of patient : koenraad
sex : male age : 27
ENTITY : infection-1
DETAILS : name infection : bacteremia
date first appearance : 12june1983
ENTITY : culture-1
DETAILS : name culture : armpit
date obtained : 15september1983
ENTITY : organism-1

DIAGNOSIS OF GENUS
value = staph
CF = 490

yes
! ?- start_session.

```
*****  
*                               *  
*   Welcome to MYCIN           *  
*                               *  
*****
```

(1) have you been able to obtain
cultures from a site at which the patient
has an infection

? yes

patient-2
(2) patient name :
? john

(3) sex :
? male

(4) age :
? 21

infection-2
(5) what is the infection
? bacteremia

(6) please give the date when this
infection first appeared (da/mo/yr)
? 12oct1981

The most recent culture associated with infection-2
will be referred to as :

culture-2
(7) from what site was the specimen for this

```

underlying_possibility_set([p(Element,Possibility)|RestFuzzySet],
                           [Possibility|RestSet]) :-
    underlying_possibility_set(RestFuzzySet,RestSet).

% funion(FuzzySet1,FuzzySet2,ResultFuzzySet): defined using the
% definition by Zadeh:
% The grade of membership of an element X in the union is the
% maximum of the grades of membership of that element X in the two
% individual sets.

funion(FuzzySet1,FuzzySet2,ResultFuzzySet) :-
    underlying_set(FuzzySet1,Set1),
    underlying_set(FuzzySet2,Set2),
    union(Set1,Set2,ResultSet),
    reconstruct_union(ResultSet,FuzzySet1,
                      FuzzySet2,ResultFuzzySet).

% reconstruct_union(ResultSet,FuzzySet1,FuzzySet2,ResultFuzzySet)
% will reconstruct the union fuzzyset by taking as the grade
% of membership the maximum of the individual grades of membership

reconstruct_union([],_,_,[]).
reconstruct_union([H|T],FuzzySet1,FuzzySet2,
                  [p(H,Possibility)|RestFuzzySet]) :-
    fmember(H,P1,FuzzySet1),
    fmember(H,P2,FuzzySet2),
    max(P1,P2,Possibility),
    reconstruct_union(T,FuzzySet1,FuzzySet2,RestFuzzySet).

% The code for fintersection is every similar to funion
% fintersection(FuzzySet1,FuzzySet2,ResultFuzzySet) is
% defined using the traditional definition by Zadeh:
% The possibility of an element X in the intersection is the
% minimum of the grades of membership of that element X in the two
% individual sets.
% We take the traditional set intersection of the two underlying
% sets as we do not want to have element with possibility 0
% in our result fuzzy set

fintersection(FuzzySet1,FuzzySet2,ResultFuzzySet) :-
    underlying_set(FuzzySet1,Set1),
    underlying_set(FuzzySet2,Set2),
    intersect(Set1,Set2,ResultSet), % traditional set intersect
    reconstruct_intersection(ResultSet,FuzzySet1,
                             FuzzySet2,ResultFuzzySet).

% reconstruct_intersection(ResultSet,FuzzySet1,
% FuzzySet2,ResultFuzzySet)
% will reconstruct the intersection fuzzy set by taking as the
% possibility the minimum of the individual grades of membership

```

APPENDIX 5
PROLOG IMPLEMENTATION OF FUZZY SET OPERATORS

```

% Filename: fuzset.pl
% Author  : Koenraad Lecot
% Date   : March 01, 1984
% Subject : Prolog Implementation of Fuzzy Sets

% Now in startup file, called "start2"
% :- ['/p/i/koen/prologlib/setutil.pl']. % load the set package

% A fuzzy set {x/a,y/b,...} will be represented as a Prolog list
% [p(a,x),p(b,y),... ]
% Elements with a grade of membership are not listed

is_fuzzy_set([]). % trivial case, the empty fuzzy set
is_fuzzy_set([H|T]) :-
    H =.. [p,Element,Possibility],
    not ( Possibility = 0 ),
    not ( fmember(Element,_,T) ),
    is_fuzzy_set(T).

% fmember( Element,Possibility,FuzzySet): Element is a member
% of FuzzySet with a possibility of Possibility
% Note that unlike the membership function for traditional sets,
% the "fmember" function never fails

fmember(Element,0,[]) :- nonvar(Element),!,fail.
fmember(Element,0,FuzzySet) :-
    nonvar(Element),
    not(member(p(Element,_),FuzzySet)).

fmember(Element,Possibility,[p(Element,Possibility)|_]).
fmember(Element,Possibility,[_|Rest]) :-
    fmember(Element,Possibility,Rest).

% underlying_set(FuzzySet,Set) will find the Set of elements of
% a fuzzy set

underlying_set([],[]).
underlying_set([p(Element,Possibility)|RestFuzzySet],
    [Element|RestSet]) :-
    underlying_set(RestFuzzySet,RestSet).

% underlying_possibility_set(FuzzySet,Set): find the
% Set of possibilities of a fuzzy set

underlying_possibility_set([],[]).

```

```

        delete_element(p(Element, _), FuzzySet1, FuzzySet2).
        % delete_element: the traditional set operation

% fadd_set_elements may be used to add a set of elements to
% fuzzyset with the same possibility ( grade of membership )

fadd_set_elements([], _, FuzzySet, FuzzySet).
fadd_set_elements([H|T], Possibility, FuzzySet1, FuzzySet2):-
    fadd_element(H, Possibility, FuzzySet1, FuzzySet3),
    fadd_set_elements(T, Possibility, FuzzySet3, FuzzySet2).

% concentration is a fuzzy set operation that squares all
% the grades of membership

con([], []).
con([p(Element, Possibility)|RestFuzzySet],
    [p(Element, Square)|RestCon]):-
    Square is Possibility * Possibility,
    con(RestFuzzySet, RestCon).

% dilation is a fuzzy set operation that takes the
% square root of all the grades of membership

dil([], []).
dil([p(Element, Possibility)|RestFuzzySet],
    [p(Element, Root)|RestCon]):-
    Root is sqrt(Possibility),
    dil(RestFuzzySet, RestCon).

% normalization is a fuzzy set operation that divides each
% grade of membership by the maximum of the grades of membership

norm(FuzzySet, NewFuzzySet):-
    underlying_possibility_set(FuzzySet, SetofPossibilities),
    maximum_list(SetofPossibilities, Maximum),
    do_norm(FuzzySet, Maximum, NewFuzzySet).

do_norm([], _, []).
do_norm([p(Element, Possibility)|RestFuzzySet], Maximum,
    [p(Element, Division)|RestCon]):-
    Division is Possibility / Maximum, % i.e. division of reals
    do_norm(RestFuzzySet, Maximum, RestCon).

% intensification

int([], []).
int([p(Element, Possibility)|RestFuzzySet],
    [p(Element, NewPossibility)|RestResult]):-
    intensify(Possibility, NewPossibility),
    int(RestFuzzySet, RestResult).

```

```

reconstruct_intersection([],_,_,[]).
reconstruct_intersection([H|T],FuzzySet1,FuzzySet2,
    [p(H,Possibility)|RestFuzzySet]):-
    fmember(H,P1,FuzzySet1),
    fmember(H,P2,FuzzySet2),
    min(P1,P2,Possibility),
    reconstruct_intersection(T,FuzzySet1,FuzzySet2,RestFuzzySet).

% The complement of a FuzzySet in a given Universe
% complement(FuzzySet1,Universe,ComplementFuzzySet) is defined:
% The grade of membership of an element in the complement is
% 1 - grade of membership in the original set
% Unlike the union and fintersection operations, the complement
% action is an operation that needs to know the given universe

complement(FuzzySet,Universe,Complement):-
    % 1. Consider the elements of the given universe, that are not
    % in the given fuzzy set
    underlying_set(FuzzySet,Set),
    subset(Set,Universe), % some checking here..
    subtract(Universe,Set,Set1),
    fadd_set_elements(Set1,1,[],FuzzySet1),
    % 2. Now we consider the elements of the given fuzzy set
    % by completing their given possibility and excluding those
    % with a grade of membership, i.e. possibility, equal to 1
    do_complement(FuzzySet,FuzzySet2),
    union(FuzzySet1,FuzzySet2,Complement). % not fuzzy union

do_complement([],[]).
do_complement([p(Element,1)|Rest],RestComplement):-
    do_complement(Rest,RestComplement). % these are not included
do_complement([p(Element,Possibility)|Rest],
    [p(Element,NewPossibility)|RestComplement]):-
    NewPossibility is 1 - Possibility,
    do_complement(Rest,RestComplement).

% fadd_element(Element,Possibility,FuzzySet1,FuzzySet2)
% add an Element with Possibility to FuzzySet1 resulting in
% FuzzySet2

fadd_element(Element,Possibility,FuzzySet1,FuzzySet2):-
    add_element(p(Element,Possibility),FuzzySet1,FuzzySet2).
% add_element: the traditional set operation

% fdelete_element(Element,FuzzySet1,FuzzySet2)
% deletes an Element to FuzzySet1 resulting in
% FuzzySet2
% ( note that we do not have a Possibility parameter here
% as elements in a fuzzy set are unique

fdelete_element(Element,FuzzySet1,FuzzySet2):-

```

APPENDIX 6
PROLOG IMPLEMENTATION OF A FUZZY DATABASE

```

% filename : fril.pl
% author   : Koenraad Lecot
% date     : April 18, 1984
% purpose  : Prolog implementation of FRIL
%
% FRIL = Fuzzy Relational Inference Language
%
%           Baldwin, Proceedings IFAC 1983

% Each relation has an additional argument :
% the possibility, of the tuple in the relation

% base relations

likes( jim,    irene,    1).
likes( john,   heather, 0.7).
likes( john,   mary,     0.6).
likes( harry,  jill,     0.4).
likes( jill,   tom,      0.2).
likes( irene,  jim,      0.9).
likes( heather, john,    0.8).

% tall is an I-type relation, this is
% a relation that takes linear interpolation values

tall( 5.25, 0).
tall( 5.50, 0.6).
tall( 5.75, 0.8).
tall( 6, 1).
tall( 7, 1).

% linear interpolation between CONSECUTIVE values
% we rely on the fact that the interpolation values are ordered
% in the base table

% interpolation

tall(X,D) :- nonvar(X),var(D),not clause(tall(X,_),true),
             clause(tall(X2,Y2),true),nonvar(X2),X < X2,
             clause(tall(X1,Y1),true),nonvar(X1),X1 < X2, X1 < X,
             check(X1,X),
             D is ( ( (Y2-Y1)/(X2-X1) ) * ( X - X1 ) ) + Y1,!.

% left extrapolation

```



```

intensify(Possibility,NewPossibility):-
    0 =< Possibility,
    Possibility =< 0.5,
    NewPossibility is 2 * ( Possibility + Possibility ).

intensify(Possibility,NewPossibility):-
    0.5 < Possibility,
    Possibility =< 1.0,
    NewPossibility is 1 - 2 *
        (1 - Possibility) + (1 - Possibility).

% utilities

max(X,Y,Y) :- Y > X.
max(X,Y,X). % else

min(X,Y,Y) :- Y < X.
min(X,Y,X). % else

minimum_list([X],X).
minimum_list([X|Y],Z) :- minimum_list(Y,U),min(X,U,Z).

maximum_list([X],X).
maximum_list([X|Y],Z) :- maximum_list(Y,U),max(X,U,Z).

setof0(X,G,S):- setof(X,G,S).
setof0(X,G,[]).

```



```

tall(X,D) :- nonvar(X), var(D), not(
    clause(tall(X,_),true)),
    clause(tall(X1,Y1),true),nonvar(X1),X < X1,
    D = Y1,!.

% right extrapolation

tall(X,1) :- not(
    clause(tall(X,_),true)),tall(X,Y),!,Y = 1,!.

check(X1,X) :- tall(Y,_),nonvar(Y),
    X1 < Y,
    Y < X,
    !,fail.

check(X1,X).

% data about persons

persons( jim, 6.10, 12.00, 1).
persons( john, 5.90, 11.90, 1).
persons( irene, 5.50, 10.00, 1).
persons( heather, 5.60, 9.60, 1).
persons( mary, 5.30, 8.50, 1).
persons( jill, 5.70, 9.20, 1).
persons( tom, 6.00, 13.50, 1).

% virtual relations

friends(X,Y) :- likes(X,Y,_),likes(Y,X,_).

possible_athlete(X) :- persons(X,Y,_,_),tall(Y,1).

has_good_friends(X) :- friends(X,Y),possible_athlete(Y).

```