723

VARIABLE PRECISION LOGIC: Hierarchical Censored Production Rules System

646

Dissertation submitted to the Jawaharlal Nehru University
in partial fulfilment of the requirements for
the award of the Degree of

MASTER OF TECHNOLOGY

in

Computer Science and Technology

by

N. K. JAIN

School of Computer and Systems Sciences
Jawaharlal Nehru University
New Delhi-110067
January 1989

T O   M Y   P A R E N T S

C E R T I F I C A T E

This work, embodied in the dissertation titled,

VARIABLE PRECISION LOGIC: Hierarchical

Censored Production Rules System.

has been done by Mr. NAVEEN KUMAR JAIN, a bonafide student of School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi.

This work is original and has not been submitted for any degree or diploma in any other university or institute.

K K Bharadwaj

Dr. K.K. Bharadwaj

Associate Professor

School of Computer and Systems Sciences

Jawaharlal Nehru University

New Delhi.

[illegible]

Prof. Karmeshu

Dean, School of Computer and Systems Sciences

Jawaharlal Nehru University

New Delhi.

ACKNOWLEDGEMENT

I am grateful to my supervisor Dr. K.K. Bharadwaj who got me interested in the field of knowledge representation and advised me to work in this particular area. His keen interest, inspiring guidance and constant encouragement through out the course of this work will be unforgettable for me.

I offer my sincere thanks to Mr. A. Tripathy, MCA student who helped me a lot by pointing out mistakes in typing and improving the report.

I feel indebted to my friend Mr. S. Nigam, for his valuable suggestions and encouragement. I am thankful to Prof. Karmeshu (Dean SC&SS) for all his help.

Thanks are also due to all the people who directly or indirectly helped me in the course of my work.

[illegible]

NAVEEN KUMAR JAIN

CONTENTS

CHAPTER 1

INTRODUCTION

R.S. Michalski and P.H. Winston are pioneers in the field of variable precision logic. In the real world, both human and computer often has to reason using insufficient, incomplete or tentative premises. Moreover, both are subject to constraints of time and memory. Variable precision logic is concerned with problems of reasoning with incomplete information and resource constraints. It offers mechanism for handling trade-offs between the precision of inferences and computational efficiency of deriving them. Two aspects of precision are specificity and certainity.

The terms specificity and certainity are used in ways that are reminiscent of the use of the term precision and accuracy in measurement theory. Precision in measurement theory represents the number of significant digits associated with a quantity, while in variable precision logic specificity is opposite of generality. Moreover, a general concept is included in a specific concept; but, a general concept may or may not include

the specific concept, e.g., a concept of triangle is more specific than a concept of polygon and a triangle must be a polygon but, a polygon may or may not be a triangle.
The term accuracy in measurement theory represents the numerical difference between the measured value and the true value of a quantity, while certainity in variable precision logic is a measure of confidence, that a given statement represents a true statement, e.g., the statement "John is working in the yard" would be

1. more certain if it is known that "weather is nice".

2. somewhat less certain if weather is unknown to the reasoning process.

3. very less certain if it is known that "weather is bad".

Winston and Michalaski [1] employed censored production rules to handle the trade-offs between the certainities of various decisions and the effort needed to derive them. The system employing censored production rules handle only certainity part of the variable precision logic.

To take into account the specificity part of precision, we are employing hierarchical censored production rules or simply HCPRs. These HCPRs are censored production rules augmented with the specificity information and are written in the form "If A

Then B Unless C Specifically S," where "S" represents the information of more specific rules or conditions.
From logical viewpoint "B" and "S" are related to each other with exclusive-nor operator (xnor); it means, if condition "B" holds then the condition represented by "S" must also hold and if condition represented by "S" holds then the condition "B" must also hold. From expositive point of view the "Specifically S" part of the rule represents the applicable set of rules when "B" holds. From control point of view the condition "B" is easy to establish and must hold for the condition represented by "S". It would prove to be the best answer if resources are tight and sufficient data is not provided. Given more time and data, a more specific answer would be tried using specificity information "S".

Chapter-2 discusses some important logical systems employed to represent knowledge in a computer. The need for extending a censored production rule to exhibit variable precision of decisions is discussed and a HCPRs system of knowledge representation is suggested. Chapter-3 describes a HCPRs system of knowledge representation in detail. A general control scheme (GCS) is discussed in Chapter-4. The GCS is implemented on two knowledge-bases designed using HCPRs system of knowledge representation. These knowledge-bases are discussed in chapter-5. One knowledge-base deals with some daily life queries, whereas another is designed for biological classifications of an unknown organism. Some more observations of a HCPRs system are given in chapter-6, which includes learning and inference rules.

The last chapter includes the concluding remarks about HCPRs system of knowledge representation.

CHAPTER 2

INTRODUCTION TO LOGICAL SYSTEMS

Representing knowledge in a computer consists of setting up a correspondence between a symbolic reasoning system and the outside world. Just as no one has yet succeeded in designing a universal programming language, no one has yet produced an ideal form for representing knowledge in an AI system. The power of a method of representation can be judged by its ability to represent the complex situations precisely and also by its ability to represent the fact that two statements have something in common: thus in a geological information system "compacted limestone" and "porous limestone" should be represented as two particular forms of "limestone" and not as two unrelated substances. This second characterstic not only improves the clarity of the representation but also reduces the demands on memory because items having properties in common need to be recorded only once, in the most general form, instead of seprate entries.

The power can be judged also by the ability to deal with imprecise arguments, especially the inductive process; the latter is more difficult to represent than purely deductive processes. This brings in the idea of "common sense" reasoning, which differs from formal logic and mathematics in that it is used when a decision has to be based on incomplete information. In real life, we often have to face the fact that our knowledge is limited and draw conclusions for which we cannot give a rigorous proof but which seem to us plausible, often expressed as "reasonable".

An introduction to some of the important logical systems, to represent reasoning and decision-making process follows:

2.1 TWO VALUED LOGIC

It is a yes or no kind of logic, in which all classes are assumed to have sharply defined boundaries. So either an object is a member of a class or it is not a member of a class. For example, mortal or not-mortal, dead or alive, male or female and so forth are classes that have sharp boundaries.

2.1.1 Syllogistic Logic

The first known logician was Aristotle (384-322 b.c.), the great philospher and natural scientist. He developed much of the theory of what has come to be called syllogistic or classical logic. Syllogistic logic essentially deals with deriving the truth (or falsehood) from a philosophical argument. This type of logic is still used because it is the basis for virtually all legal argumentation.

Study this short argument.

> John is a man.
>
> All men used to be boys .
>
> Therefore, John used to be a boy.

Common sense tells us that John was a boy before he became a man. After being converted to syllogistic form, this argument becomes

```
          J --> M
    All   M --> B
  hence   J --> B
```

In many ways, syllogistic logic is simply a formalisation of common sense. However, because syllogistic logic is based in natural languages, it suffers from the inherent flaws of a natural language. Natural languages are often imprecise and can be misunderstood. Also, people tend to hear or read selectively, which can cause further confusion. This lack of precision eventually led to the invention of symbolic logic.

2.1.2 Symbolic Logic

Symbolic logic began with G. W. Leibniz(1646 - 1717), but was forgotten when he died. The entire field was rediscoverd by the person generally given credit for its invention, George Boole(1815 - 1864). This type of logic is called Boolean logic. Symbolic logic deals with the abstraction of concepts into symbols and interconnection of these symbols by certain operators.

```
example: IF   (P is true ) and
              (Q is false)
        THEN  (P  or Q is  true) and
              (P and Q is false)
```

The idea of using formal logical system to represent reasoning and decision-making process was first suggested in a paper by John McCarthy in 1958 [5].

There are two distinct but interlocking branches in symbolic logic, the first is propositional logic and the other is predicate calculas. We describe first the propositional logic.

2.1.2.1 Propositional Logic - Propositional logic deals with the determination of the truthfulness or falseness of various propositions. A proposition is a properly formed statement that is either true or false.

Although, propositional logic forms the basis for both intelligence and computer languages, one cannot use it by itself to represent human knowledge of the world, because it lacks the ability to represent relationship between objects, and it cannot be used on classifications. This shortcoming leads to predicate calculas.

2.1.2.2 Predicate Calculas - Predicate calculas sometimes called predicate logic, is simply an extention of propositional logic. The basis of predicate calculas is the predicate, which is essentially a function that returns either a value of true or false depending upon its argument.

example1: The predicate Dog defined by Dog(X) : " X is dog "
takes the value True if X=Rover and False if X=Pussy.

```
example2:  is_hard ( rock )   ==>  true,
           is_hard ( cotton ) ==> false.
```

In propositional logic these two predicates and arguments become:

a rock is hard.

cotton is not hard.

In predicate calculas, it is possible to create a function that determines the hardness of any object. The predicate calculas uses variables to generalise predicates, e.g.,
man(X) ==> Not woman(X).

Predicate calculas uses two quantifiers, existential (there exists) and universal (for all), e.g.,

All cats are animals <==> for all X Cat(X) ==> animal(X)

Every boy has a bicycle

<==>

for_all X (Boy(X) ==> there_exists Y (Bicycle(Y) and own(X, Y)))

2.2 MULTIVALUE LOGIC

A polish mathematician J. Lukasiewiecz, first developed the concept of multivalued logic during the 1920s. In multivalued logical systems, there are more than two truth values. There may be a finite or infinite number of truth values, i.e., an infinite number of degrees to which a property may be possessed. In a three valued system, for instance, something can be true, false, or on the boundary.

2.3 NONMONOTONIC LOGIC

A logic in which a conclusion stands no matter what new axioms are added, is called monotonic logic. Traditional system based upon predicate logic are monotonic in the sense that the number of statements known to be true is strictly increasing over time. Neither of new statements added to the system or new theorem proved, will ever cause a previously known or proven statement to become invalid.

Monotonic systems are not very good at dealing with incomplete information, changing situations, and generation of assumptions in the process of solving problems. Nonmonotonic logic [4] allows statements to be deleted from, as well as added to, the database. Among other things this allows the beliefs in one statement to rest on a lack of belief in some other one. Rarely does a system have at its disposal all the information that would be useful. But often when such information is lacking, there are some sensible guesses that can be made, as long as no contradictory evidence is present. The construction of these guesses are known as default reasoning. We know that one of a set of things must be true and, in the absence of complete information, we choose the most likely. Most people like flowers. Most dogs have tails. The most common color for Swedes is blond. These examples illustrates one common kind of default reasoning, which may be called most probable choice. Another important kind of default reasoning is circumscription [3], in which we assume that the only objects that can satisfy some property P, are those that can be shown to satisy it.

2.4 PROBABILISTIC LOGIC

Probabilistic logic makes it possible to represent likely but uncertain inferences. There are three types of situations in which it is tempting to use probabilistic logic:

- The relevant world is really random, for example, the motion of electrons in atom or the distribution of people who will fall ill during an epidemic.
  Example1. consider the problem of deciding which card to play in a game of bridge. We are dealing with a genuinely random world. So we will have to use probabilistic reasoning.

- The relevant world is not random given enough data but our program will not always have access to that much data, e.g., the likelihood of success of a drug at combatting a disease in a particular patient.
  Example2. consider the problem of diagnosing people's illness from clinical records. There is some randomness in our description of the world, since medical science does not completely understand how the body works. In addition we must design a program that can function even if it does not have access to all the data medical science could conceivably provide, since some clinical test are expensive and dangerous. With such incomplete data to work with, we will have to use probabilistic logic.

- The world appears to be random because we have not described it at the right level.

  Example3. consider the character recognition problem, where if, characters are viewed as a collection of dots of ink, then there appear to be a great deal of random variation in their appearence. But if they are analyzed in terms of arcs and lines, then much of the randomnesss disappears, i.e., it no longer matters how wide the lines are. For this problem, we should use as little probabilistic logic as possible.

2.5 FUZZY LOGIC

In 1965, Zadeh introduced the concept of a fuzzy set as a model of a vague fact. In every day life we often deal with imprecisely defined properties or quantities: "a few books", "a long story", "a beautiful woman", "a tall man ". The key idea in fuzzy set theory is that an element has a degree of membership in a fuzzy set. Thus a proposition need not be simply true or false, but may be partly true to any degree. We usually assume that this degree is a real number in the interval [0, 1].
Consider the fuzzy set "tall". The element are men and their degree of membership depend on their heights. For example, a man who is 5 feet tall has degree 0, a man who is 7 feet tall might have degree 1 and men with intermediate heights might have

intermediate degree.

2.6 VARIABLE PRECISION LOGIC

In 1986, Michalski, R. S. and Winston, P. H. in their paper [1], introduced the censored production rule to exhibit variable precision logic. Variable precision logic is concerned with reasoning with incomlete information and resource constraints. It offers mechanism to handle trade-offs between the precision of inferences and the computational efficiency of deriving them.
You can not tell an ordinary logic-based reasoning system much about how you want it to do its job. You can not give the following instructions, for example

- Give me a reasonable answer immediately; if there is enough time, tell me you are more confident in the answer or change your mind and give me another better answer.

- Give me a reasonable answer immediately, even if somewhat general; if there is enough time, give me a more specific answer.

- Give me a highly certain answer only, even if somewhat general; if there is enough time, give me a more specific answer.

- Give me a highly specific answer in the time allowed, even if you are less confident about it; if the allowed time is enough, tell whether you are more confident in it.

There may be various other requirements of this type, which might arise in real life. So our reasoning system should facilitate these requirements on real life problems. To understand the variable precision logic system consider the following example.

Suppose you are interested to know, what X is doing on sunday and you know that on sundays people generally do not prefer to stay inside their houses. Noticing this fact, a quick answer may be that "X is outside his house". But by considering the fact that "X has to appear in annual examination tomorrow", you have to withdraw the previous decision that "X is outside his house" and have to give another better answer that "X is inside his house". On taking in to account the fact that "weather is bad", you can answer with more certainity that "X is inside his house". A more specific answer, on taking in to consideration the fact that "X is in his reading room", would be that "he is reading for tomorrow's examination". A system that gives more specific answers, given more time is what we call a variable specificity system. A system that gives more certain answers, given more time is what we call a variable certainity system.

There can be various combinations of the two systems, reflecting the fact that specificity and certainity are inversely related.

Variable specificity and variable certainity are two aspects of variable precision. Thus, in general variable precision system is a system that exhibits either variable specificity or variable certainity or some trade-off between the two. Next consider the important production rules system, which on suitable modification will become the basis of the variable precision logic.

2.6.1 Production Rules

A production rule is a situation-action couple, meaning that whenever a certain situation is encountered, given as the left side of the rule, the action given on the right side is performed. It can be written in the form "If premise Then action." The premise is a conjunction of predicates representing certain situation and the action is what is to be done when premise is satisfied. Very often the action is the taking of some decision, then rule becomes an implicative assertion of the form:

"premise ==> decision."

But this is not always the case. There is no apriori constraint on the form of the situation or of the action. A system based on production rules will usually have three components:

1. The rule base, consisting of the set of production rules.

2. The fact bases, consisting of some useful definitions and data structures containing the known facts.

3. The interpreter of these facts and rules, which is the mechanism that decides which rule to apply and initiates the corresponding action.

The facts and the rules have a syntax that is known to the interpreter; the latter can therefore manipulate these logically, deciding on their truth or otherwise, in some programs, deriving new facts from them or suppressing certain facts. Consider this simple example:

Rule base: R1 If X is an animal and X mews then X is a cat (one rule only)

Fact base: F1 Felix is an animal

F2 Felix mews

Fact base after the interpreter has scanned both facts and rules

F1 Felix is an animal

F2 Felix mews

F3 Felix is a cat (new fact obtained by applying R1 with X = Felix)

A production rule lacks certain aspects of common sense knowledge. Precision of inferences remains constant in production rules system, since neither certainity nor specificity can vary with resource constraints. The production rules system is not suited to reasoning with incomplete information and resource constraints. Also, the production rules system of knowledge representation is not natural to rule repair mechanism. Whenever, a contradiction to a production rule is found, there are various possibilities to handle it:

1. To consider the rule invalid, and ignore it in future.

2. To continue to use the rule without change, realising that it will result in error occasionally.

3. To modify the rule, so that the rule applies correctly to all encountered situations.

4. To develop a new rule, substituting the new rule for the old.

5. to remember the situations for which the rule does not work, treating them as exceptions.

Action 1, is simple and prevents us from making errors. It deprives us of the benefit of using the rule when it does work. Action 2, is also simple. It preserves the benefit of using the rule when it does work, but using it will lead to some error. If modification to be made to a rule is small, then action 3, is the better choice. But if this modification is unclear or complicated, then Action 4, is the better choice. Both these later actions lead to a better and more precise rule, but require time and effort. In science, where standards for precision are high, one of these two actions is the usual choice. (the problem of incrementally refining rules to accomodate new facts is explored in [8].)

If exceptions are few, then Action 5, is a good choice. It preserves the usefulness of the old rule, but prevents making mistakes in situations recognised as exceptions. Even when exceptions are more than few, it is the best action to take, particularly when it is not clear how to make changes to the old rule or how to create a new one. Production rules with exceptions are called censored production rules.

2.6.2 Censored Production Rules

Winston [2] first introduced the concept of censored production rules. Censored production rules are production rules augmented with exceptions. It is of the form: "If premise Then conclusion Unless censor." The censor is a logical condition (a predicate or a disjunction of predicates) that when satisfied, blocks the rule. Thus, a censor can be viewed as a statement of exceptions to the rule.

These forms of representation are more natural and comprehensible than other equivalent logical forms. A simple rule with exceptions may be better than a complicated one without exceptions, particularly when the exceptions occur only rarely. Also, if exceptions are few, then to remember exception conditions, using censored production rule is a good choice.
Some initial work, in the direction of inductive learning of decision rules with exceptions, was done by Becker [7].

Censored rules support a number of obvious alternate control schemes. In Winston's formulation an unlimited effort is put into showing that premise is true, but only one-step effort is put into showing that censor is true, and when one-step effort fails, the censor condition is assumed to be false. Here are the two extreme possibilities of control schemes:

- The show-me method: Treat the unless operator as if they are exclusive or operators.

- The ask-question-later method: Ignore all censors.

Employing censored production rules as a vehicle to implement variable precision logic exhibits only variable certainity, whereas specificity stays constant. So there should be some other better representation schemes or some modification to the existing censored production rules representation. We prefer latter, because it would retain the advantages of a censored production rule, while exhibiting more intelligence. The resulting rule will be called a hierarchical censored production rule or simply HCPR. We will discuss it in detail, in the next chapter.

CHAPTER 3

HCPRs SYSTEM

For a reasoning process, at any state of the system, it is valuable to have information about the applicable rules, because then it need not find them using exhaustive search of the whole rule base, but on considering the given information, it can select a set of most relevant rules to the current state of the system, e.g., consider the query - "What John is doing?" ; after finding by reasoning process that "he is working in the yard," the next line of action taken by the system would be to get more specific answer, or it should apply rules which can provide decisions of the type:

- He is raking leaves.
- He is watering.
- He is shaping plants.

- He is preparing field.

rather than the rules, which gives decisions of the type:

- He is reading a story book.

- He is eating fruits.

- He is climbing on a tree.

- He is watching a movie.

because these decisions are totally unrelated to the previously inferred decision, i.e., "John is working in the yard". These out of context rules should be avoided, because they might require some irrelevant information to be provided, which surely, one would not like. Intelligent systems should also be able to discard most of the task irrelevant information quickly, and should concentrate on main line of reasoning.

Such a behaviour exhibited by system should be regarded as "intelligent behavior" since the availability of information of applicable set of rules is evidence of more complex reasoning process than a blind search through all the possibilities. One of the criteria for intelligence is the ability to deal with complexity (where complexity is necessary: recall Einstein's dictum, "Things should be made as simple as possible, but no simpler".)

Augmenting censored production rule with the information of rules, which should apply next to get a more specific decision, results in a hierarchical censored production rule or simply a HCPR. Next section describes a HCPR in detail.

3.1 HCPR

A HCPR is of the form "If A Then B Unless C Specifically S" . It is created by augmenting the censor production rules with the specificity information S . The specificity information to a rule is hint about a set of rules in a rule-base, such that:

(A) these rules are the most likely to be satisfied,

(B) these rules are the most relevant to the current state of the system and

(C) the decisions from these rules are more specific than the decision of the augmented rule.

We will employ a rule-tree as an underlying representational and computational mechanism to handle various trade-offs. Where, a rule-tree is a collection of all related HCPRs for the same domain of problems. Next section describes a rule-tree in detail.

3.2 RULE-TREE

A rule-tree is a collective and systematical representation of all related HCPRs about a given concept. From collective we mean, all rules in a knowledge-base about a particular concept. A HCPR in a rule-tree is a quanta of knowledge about a particular domain of problems. It is a quanta in the sense that it cannot be further divided into two or more simpler rules, and is complete in itself. The concept of rule-tree, provides a mechanism to systematically handle the problems in a particular domain of knowledge. Also, it provides an efficient means to handle new information, which produce either a contradiction or which cannot be explained on the basis of the current knowledge-base. A knowledge-base may contain one or more rule-tree each for different domain of problems.

The general concepts in a rule-tree are represented at relatively low level of specificity (in the vicinity of the root) and the specific concepts are represented at relatively higher level of specificity (in the vicinity of the leaves of the rule-tree). So a rule-tree is a systematic representation of HCPRs for similar domain of problems. Any subtree of a rule-tree cannot represent more general concept than rule-tree itself, or in other words, its domain of problems is relatively restricted. A general rule is one, on which one or more specialised rules are dependent directly or indirectly, i.e., leaves of a rule-tree represent the most specialised rules and nodes other than leaves represent relatively more general rules. Rules dependent on same immediate general rule are called sibling rules and the general rule is called the parent rule.

For simplicity, from this point onwards, we will assume that sibling rules in a rule-tree will give mutually exclusive decisions. This mutually exclusive property between sibling rules would be made explicit in the specificity information part of the parent rule, using "xor" operator.

1. The rule-tree offers mechanisms to handle various trade-offs.

   i.e., (a) it offers mechanism to handle trade-offs between the precision of inferences and computational efficiency of deriving them. (b) it offers mechanism to handle trade-offs between the certainity of conclusion and its specificity.

2. The representation using rule-tree facilitates efficient use of memory, e.g., consider the query "Did John hit Peter?" , issued to an ordinary logic based reasoning system having single fact that "John punched Peter." On checking the fact-base, it will reply "no" , and if another fact that "John hit Peter" is included in the system then it will reply "yes", to the above query. Though, actions hit and punched are related to each

other, but it has to store separate facts for these related actions of "hit" and "punched". So it is not an efficient utilisation of memory. Now consider the following knowledge base:

```
> rule-tree:
     R1:  strike (X, Y) ==> hit (X, Y) $
                                              (punched xor
                                               slapped xor
                                               kicked)


          hit (X, Y) and strike_by_fist (X, Y)
                                            ==> punched (X, Y).
          hit (X, Y) and strike_by_hand (X, Y)
                                            ==> slapped (X, Y).
          hit (X, Y) and strike_by_foot (X, Y)
                                            ==>  kicked (X, Y).
   R2:  strike $
                  (strike_by_fist xor
                   strike_by_hand xor
                   strike_by_foot)
> fact-base:
   F1:  Strike_by_hand(John, Mary).
   F2:  Strike_by_foot(John, Jim).
   F3:  Strike_by_fist(Jim,  Peter).
   F5:  Strike_by_fist(John, Peter).
```

From a rule-tree a general concept can be inferred if its specific concept is given in the fact-base. So in HCPRs system only the most specific facts are needed to be stored in the fact-base, such that, all the general concepts may be inferred from the rule-tree and the fact-base, i.e., using a single fact F1, rule-tree R1 and R2 it can answer the following questions:

- From fact F1: strike_by_hand(John, Mary)
  it would reply yes to the query - Did John strike Mary by hand ?.

- From F1, R2; it results in a new fact RF1 -
  RF1: John struck Mary.

- From R1, RF1; it gives a new fact RF2 -
  RF2: John hit Mary ; using RF2 it is able to answer the query - Did John hit Mary ? and reply yes.

- From F1, RF2 and R1; it gives a new fact RF3 -
  RF3: John slapped Mary.

- The mutually exclusive property of sibling rules decisions namely - punched, slapped and kicked (these are mutually exclusive, since at any instant of time only one action could be performed) , a rule-tree R1 and a fact RF3 results in the following facts (negative).

NF1: John did not punch Mary.

NF2: John did not kick Mary.

NF3: John did not strike by fist.

NF4: John did not strike by foot.

So using little knowledge stored, it is able to answer a number of queries.

3. This rule-tree provides an efficient mechanism to discard most of the task irrelevant information provided to an intelligent system. It considers or asks task relevant information only (it may be noticed from the number of example sessions in Appendix-B and Appendix-C).

4. The specificity information provides a systematic way to proceed for the conclusion. It discards a large number of rules at each level of specificity of conclusion, thus makes possible the most rapid progress to a useful conclusion. So to find a conclusion, inference engine is required to consider only a small set of rules (information about which is given by the last successful rule) and hence, it is not required to consider the whole knowledge-base. To understand a forward chaining using a rule-tree, consider the rule-tree R1 for the general concept of " hit" (page 26). Suppose a robot with rule-tree R1 is watching John and Mary in action.

On seeing that John struck Mary by some object, it will infer that "John hit Mary". His next line of action would be to see, whether the object was his fist (because it comes first in the specificity information) and on finding that object was his hand (a relatively finer observation), it will infer that "John slapped Mary".

So it is superior to other systems because it requests information that has the greatest importance (or impact), given the current state of the system. The general theory of operation is that the system requests as its next piece of information the one that will remove most of the uncertainity from the system, e.g., The doctor first asks if the child has fever because the answer to this question narrows the greatest number of possibilities. If your answer is "yes" to the first question, then the doctor asks you if your child is nauseated. As with the first question, the doctor asks this question over other questions because its answer has the greatest impact given the current state. This procedure continues until the doctor can make a diagnosis. In this example the key point is that doctor selects each question to make the most rapid progress to a conclusion.

At this point, a formalisation of rule-tree is required. Since, a rule-tree is simply a systematic representation of HCPRs, a formalisation of a HCPR at its ith level of specificity is given.

3.2.1 Formalisation

Following is a HCPR at ith level of specificity of a rule-tree:

```
A(j0,j1,j2.....j(i-1)): YA(j0,j1,.....,j(i-1)) and
B(j0,j1,......ji,1): YB(j0,j1,......,ji,1) and .............and
B(j0,j1,....ji,p(j0,j1,...ji)): YB(j0,j1,....ji,p(j0,j1,....ji))
          ==>              A(j0,j1,....ji): CF(j0,j1,....,ji)     @
                           X(j0,j1,....ji,1): CF(j0,j1,..,ji,1)    or
             X(j0,j1,....,ji,2): CF(j0,j1,.....,ji,2) or.......or
             X(j0,..,ji,m(j0,j1,...ji)): CF(j0,j1,..m(j0,j1,..ji))
              : YA(j0,j1,....,ji)   $
             A(j0,j1,....,ji,1)  xor
             A(j0,j1,....,ji,2)  xor..... xor
                                   A(j0,j1,....,ji,n(j0,j1,...,ji))
```

Where, A(j0,j1.....j(i-1)) is a decision derived from parent rule of A(j0,j1.....j(i-1),ji) at (i-1)th level of specificity, and A(j0,j1...ji,1); A(j0,j1...ji,2); .... are its child rules decisions at (i+1)th level of specificity. If a rule does not have any child rule then it represents a leaf-rule of the rule-tree, and if it does not have any parent rule then it is the root-rule of the rule-tree.

1. The symbols "@" and "$" are for "Unless" and "Specifically" respectively. If resources are tight then reasoning process can neglect information associated with these symbols completely or selectively.

2. Nm is the total number of levels of specificity in the rule-tree and i is a positive integer less than Nm.

3. j0 is always zero, such that A(j0) <==> A(0) <==> A. It shows that at 0th level of a rule-tree only one rule is there. In other words, each tree has only one root.

4. j1,j2,...ji are non-zero positive integers. Their values decide the particular rule at ith level of specificity of a rule-tree.

5. p(j0,j1,...ji) is a positive integer. It gives the number of predicates other than the decision from its parent rule, using which A(j0,j1,..ji) might be inferred.

6. m(j0,j1,j2,..ji) is a positive integer. It gives the number of exception conditions to the rule.

7. n(j0,j1,j2...ji) is a positive integer. It gives the number of child rules associated to the HCPR. In other words, it gives the numbers of rules which reasoning process would tried next if assertion A(j0,j1,....ji) holds.

8. YA(j0,j1,...,j(i-1)) ; YB(j0,j1,..,ji,1) ; .... YB(j0,j1,....ji,p(j0,j1,...ji)) and YA(j0,j1,...ji) are variable certainity factors assigned to various predicates during execution. Their values depend very much on resource constraints and input data. This is related to reasoning with tentative premise.

9. YA(j0,j1,j2,..,ji) is the certainity factor with which the decision A(j0,j1,j2,...,ji) is inferred.

There are several ways to combine certainity factor at different levels of specificity depending on the type of knowledge-base system. Our knowledge-bases (designed for implementation part) employed the following method to calculate the certainity factor of a decision.

YA(...) = (Strength of premise) *

(Strength of implication)

More clearly,

$$
\begin{aligned}
YA(j0,j1,...ji) = \min [& YA(j0,j1,...,j(i-1)) , \\
& YB(j0,j1,...ji,1)..........., \\
& YB(j0,j1,..ji,p(j0,j1.....,ji))] * \\
& [CF(j0,j1,..ji) + \Sigma(n)\ CF(j0,..ji,n)]
\end{aligned}
$$

Where $\Sigma(n)$ represents summation over n, and n is a positive integer such that, X(j0,j1,..ji,n) is the nth exception condition to the rule and it is found to be false. If some nth censor condition X(j0,j1,...ji,n) is found to be true then YA(j0,j1,...ji) = 0.0, or A(j0,j1,...ji) is false. Also, if for some value of n exception X(j0,j1,..ji,n) is unknown then CF(j0,..ji,n) is 0.0, i.e., it would contribute nothing to the confidence, in deriving a decision.

10. CF(j0,j1,...ji) ; CF(j0,j1,...ji,1)................. and CF(j0,j1,....,m(j0,j1,...,ji)) are constant certainity factors. CF(j0,j1,...ji) is 0-level strength of implication and it should always be greater than 0.50. CF(j0,j1,j2...ji) + $\Sigma(n)$ CF(j0,j1,.....ji,n) is 1-level strength of implication, where n ranges from 1 to m(j0,j1,...ji). CF(j0,j1,j2......ji,1)......and CF(j0,j1.....m(j0,j1,.....ji)) are numerical estimates of likelihood of 1st, 2nd, ...and m(j0,j1....ji)th exception condition respectively. The 1-level strength

of implication can not exceed 1.0 . The values to these constants should be given by either experts in that area of knowledge or using probability theory.

3.2.2 Rule-tree is a tree of Decisions

For example, consider the following general rule-tree

```
                               A(0)
                              /    \
                            /         \
                          /             \
                      A(0,1)         A(0,2)
                    /    |           /    |  \
                  /      |        /       |    \
                /        |      /         |      \
           A(0,1,1) A(0,1,2)  A(0,2,1) A(0,2,2) A(0,2,3)
                               /   |  \              |
                             /     |    \            |
                           /       |      \          |
                          /        |        \        |
                 A(0,2,1,1) A(0,2,1,2) A(0,2,1,3)  A(0,2,3,1)
                 /   |                 /    |
               /     |              /       |
             /       |             /        |
           /         |           /          |
A(0,2,1,1,1) A(0,2,1,1,2) A(0,2,1,3,1) A(0,2,1,3,2)
        /                               /   |     \
       /                              /     |       \
      /                             /       |         \
     /                            /         |           \
A(0,2,1,1,1,1)       A(0,2,1,3,2,1) A(0,2,1,3,2,2) A(0,2,1,3,2,3)
                                    /     |       \
                                  /       |         \
                                /         |           \
                              /           |             \
               A(0,2,1,3,2,2,1) A(0,2,1,3,2,2,2) A(0,2,1,3,2,2,3)
```

Fig.- 1

Each rule in a rule-tree is of the form "If premise Then decision Unless censor_conditions Specifically specificity_information". The nodes in rule-tree (fig.1), represent the decisions of increasing specificity from root to leaves and links between nodes represent the specificity information. The premises and censor-conditions are implicitly assumed with the nodes. This rule-tree has grown up to the sixth level of specificity (Nm = 7). Nodes at different levels of rule-tree denote the decisions with the specificity of that level. Decisions of higher specificity depend on the existance of their ancestor rules decisions. A(0,2,1,3,2) is a decision at 4th level of specificity, A(0,2,1,3) is its parent rule decision at 3rd level of specificity and A(0,2,1,3,2,1); A(0,2,1,3,2,2); A(0,2,1,3,2,3); are its child rules decisions at 5th level of specificity. The values of integers n(0,2,1,3,2,2,2), n(0,2,1,3,2,2), n(0,2,1,3,2), n(0,2,1,3), n(0,2,1), n(0,2) and n(0) in this particular rule-tree are 0, 3, 3, 2, 3, 3 and 2 respectively.

Consider a particular example of a rule-tree for the concept of plane figure (fig.2), to understand a rule-tree.

```
                              plane figure
                            /              \
                          /                  \
                      convex                   non convex
                    /        \
                  /            \
                /                \
          polygon                 ovalfigure
          /      \                  |      \
        /          \                |        \
   irregular        regular        ellipse   circle
                   /   |   \
                 /     |     \
               /       |       \
      equilateral    square    pentagon
       triangle
```

Fig.- 2

This rule-tree for plane figures, represents the plan to identify an unknown plane figure. The root of the tree represents the general concept of plane figure, and its subtree with root convex represents the somewhat less general concept of convex figure. This subtree includes all the specialised rules relevant to the general concept of convex figure, but no rule about the concept of non-convex figures.

This rule-tree will infer a given figure is triangle only after it has inferred that it is a polygon, convex and plane figure.

CHAPTER 4

CONTROL SCHEMES

Consider a HCPR: A ==> B @ C

$ D.

Like exception conditions, specificity information "D", to a rule may be incomplete and in such case, "D" should be interpreted as - (D1 xor D2 xor ... xor Dn xor Unknown), instead of (D1 xor D2 .....xor Dn) when it is complete.

There are different viewpoints of "unless" operator "@" and "specificity" operator "$" namely - logical, expositive and control viewpoint.

1. from a logical viewpoint, the "unless" operator "@" between "B" and "C" acts as the "xor" operator and "specificity" operator "$" between "B" and "D" acts as the "xnor" operator.

2. From an expository viewpoint:

   - the "A ==> B" part of a HCPR expresses an important information, while the "@ C" part acts only as a switch that changes the polarity of "B" to "not B" when "C" holds.

   - the "A ==> B" part of a HCPR expresses the general part of information which requires compartively less efforts. If "B" holds then control is passed to the set of implications given by "D". The "$ D" part gives the more specific information part of the rule.
     The expositive aspect of operator "$" gives that the "$ D" part of the rule, will require more data (or fine observation) than "A ==> B" part of the rule, e.g., the observation "John struck Mary using his fist" is more fine than the observation "John struck Mary". The latter observation requires less efforts, but the first one includes the latter observation too.

3. From control viewpoint HCPRs are intended for situations in which

- the implication "A ==> B" holds frequently and the assertion "C" holds rarely. Systems employing HCPRs are free to ignore exception conditions when resources are tight. Given more time, the exception conditions are examined, lending credibility to high speed answers or changing them.

- the assertion "B" is more general than assertions given by "D" and it is a must for holding of assertions given by "D". Also, the assertion "B" would prove to be the best answer, if sufficient resources are not available to find the assertions given by "D". Systems using HCPRs are free to ignore the specificity information provided by "D". If more resources are made available to the reasoning process, it will try to establish assertions given by "D" also, and would give a more specific answer.

Therefore, the HCPRs system exhibits variable precision of conclusions, reflecting variable investment of computational resources in conducting reasoning. Next, we consider a General Control Scheme (GCS), which may be employed in reasoning process of a HCPRs system of knowledge representation.

4.1 A GENERAL CONTROL SCHEME

First we define a threshold certainity factor m, which separates the CF of true and false answers. Such that, answers with CF >= m are considered true and answers with CF < m would be considered false. By default, the value of m is 0.5, i.e., answers with CF < 0.5 are false and answers with CF >= 0.5 are true. It may vary between 0.0 and 1.0 (exclusively) depending on the requirement, (0 < m < 1.0).
The control aspect of "specificity" and "unless" operator supports various obvious control schemes. There are two extreme cases of these control schemes.

4.1.1 Control Scheme 1 (CS1)

Under this control scheme for each level of specificity treat the "unless" operators as if they are exclusive-or operators. It ignores the expectation information of "unless" operator. This is good in situations in which

1. expectation conditions are unreliable.

2. nothing should be assumed.

3. there is no resource constraint.

The fig.1 is for CS1 where, m varies between 0.09 to 0.99 in the steps of 0.20 and parameters e and k remain constant.

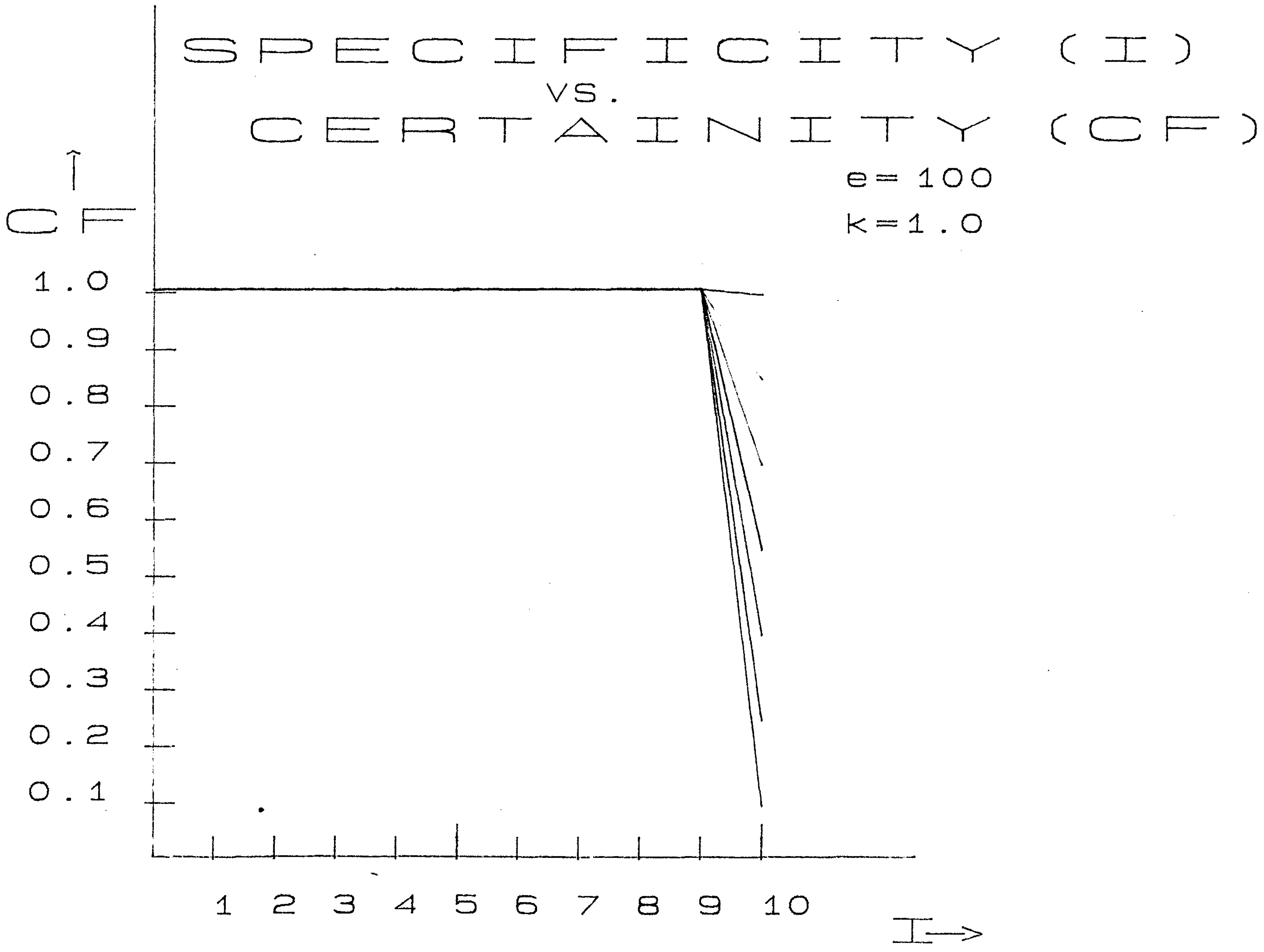
SPECIFICITY (I)
VS.
CERTAINITY (CF)
e= 100
k=1.0
CF
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
1
2
3
4
5
6
7
8
9
10
I

fig..

4.1.2 Control Scheme 2 (CS2)

This control scheme ignores all censor conditions wherever possible. This is good in situations in which

1. rapid response is critical.
2. resources are very tight.
3. censors to the rules are rare and unlikely to occur.

In CS2, censor conditions would be considered only if certainity factor of the answer is less than m.
The fig.2 is for CS2 where, m varies between 0.09 to 0.99 in the steps of 0.20 and parameters e and k remain constant.

4.1.3 General Control Scheme (GCS)

The censor conditions and specificity informations support various control schemes. CS1 and CS2 are two extreme cases of the various control schemes possible. Here we propose a general control scheme GCS, using which any of the possible control schemes may be generated on providing the control parameters e, m and k (fig.1 to fig.6).

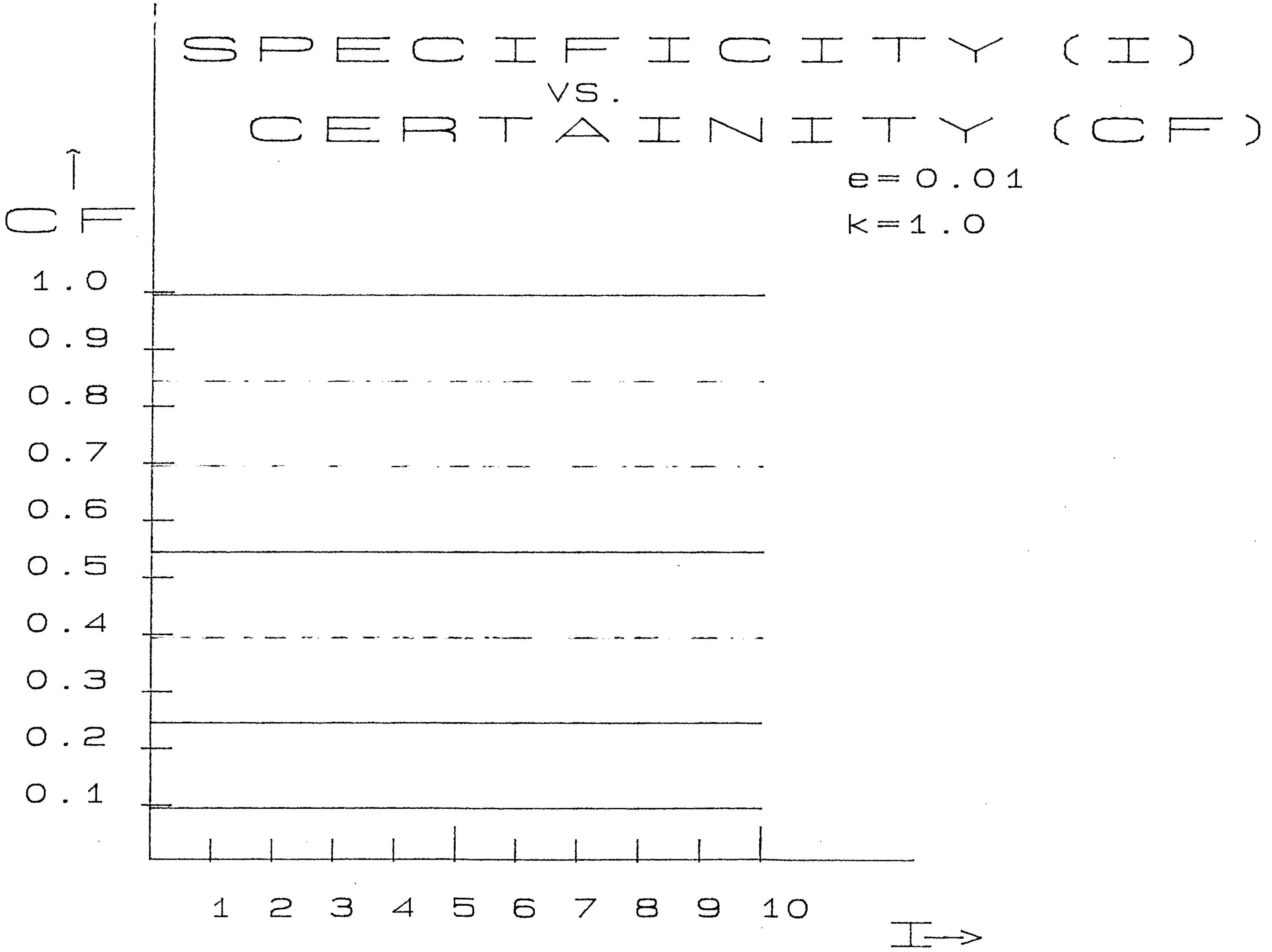
SPECIFICITY (I)
VS.
CERTAINITY (CF)
e=0.01
k=1.0
CF
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
1
2
3
4
5
6
7
8
9
10
I

fig..

> GCS:

```
1.  N = Ceiling ( Nm * ( expt (1.0 - m) (1.0 / (e * k))))
2.  CF(I) = 1.0 - (1.0 - m)*(expt ((I + 1) / N)) (e / k))
                                          : 0 <= I < N
3.  CF(I) = m  : N <= I < Nm, N < Nm.
```

Where, Nm is the total number of levels of specificity (Nm > 0) of the rule-tree to which the query is issued.

N is the number of lower levels of specificity (N <= Nm) for which control schemes in between CS1 and CS2 (inclusively) would be employed.

Nm-N is the number of higher levels of specificity for which CS2 would be employed.

CF(I) is the requirement on CF at the ith level of specificity by the particular control scheme employed. Its value may vary between m and 1.0 (inclusively) or m <= CF(I) <= 1.0 .

Parameter e is a real number and depends only on the resource constraints. It may vary between 0.0 and infinity (exclusively).

In fig.3 to fig.6 each, value of parameter e varies between 0.25 to 2.10 in the steps of 0.30.

The following are various ranges of e, showing different resource constraints:

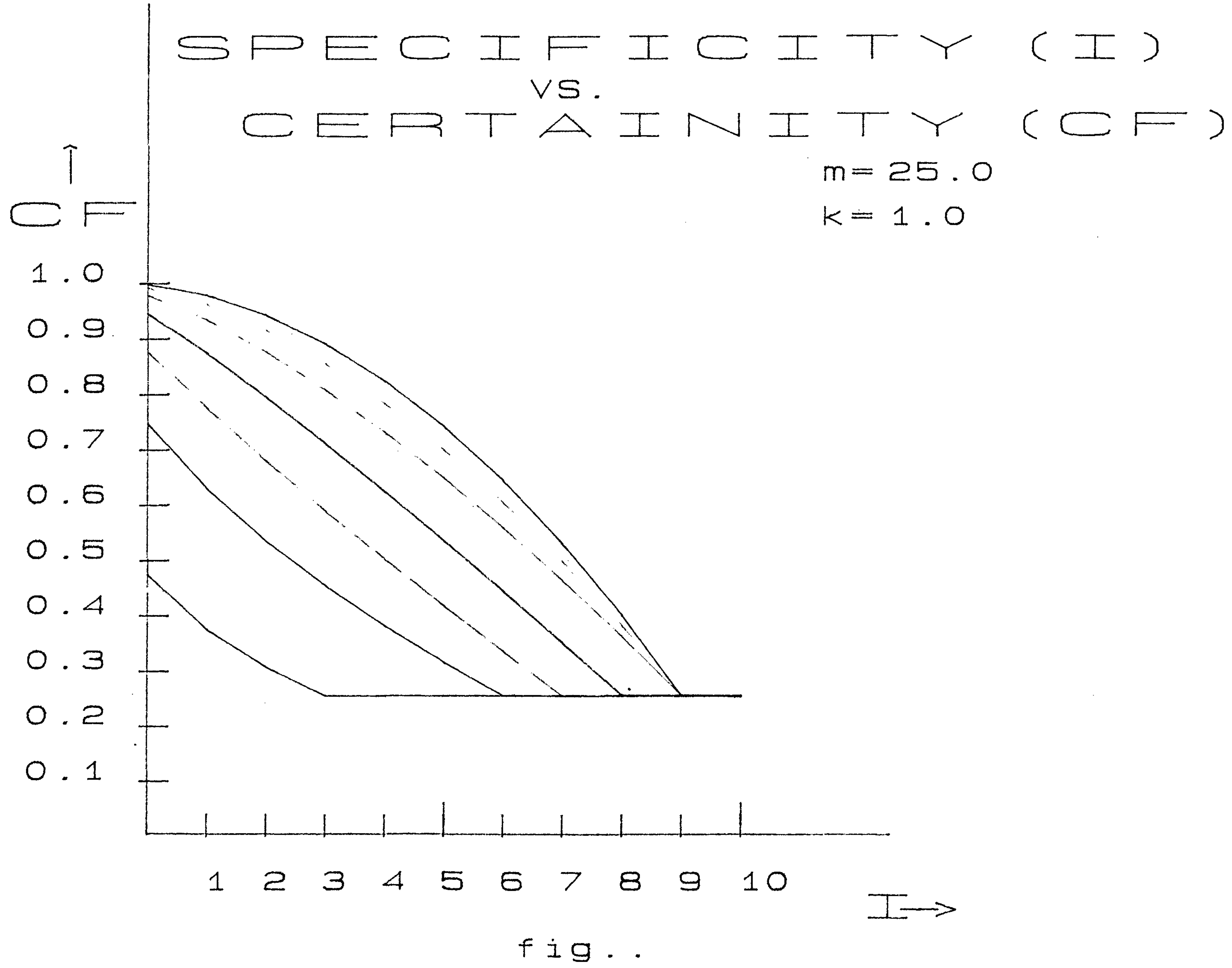
SPECIFICITY (I)
vs.
CERTAINITY (CF)
m= 25.0
k= 1.0
CF
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
1
2
3
4
5
6
7
8
9
10
I

fig..

A. very-2 high resource constraint: $0.125 <= e < 0.250$

B. very high resource constraint: $0.250 <= e < 0.50$

C. high resource constraint: $0.50 <= e < 1.0$

D. moderate resource constraint: $e = 1.0$

E. low resource constraint: $1.0 <= e < 2.0$

F. very low resource constraint: $2.0 <= e < 4.0$

G. very-2 low resource constraint: $4.0 <= e < 8.0$

The default value of parameter e is 1.0, which shows that reasoning system always has some constraints of time or memory or both. The value of e is dependent on real resource constraints (allowed time and memory), which in turn are dependent on particular type of knowledge-base, i.e., for some type of problems, time of 30 minutes may be reasonable, but for others time of 30 mili-seconds is more than enough.

Parameter k is a real number, which for given values of parameters e and m, will decide:

- the number of higher levels Nm-N, for which CS2 is to be employed

- the number of lower levels for which CS1 is to be employed and

- the number of in between levels for which control scheme between CS1 and CS2 would be employed.

(1) if $k > 1$ then Nm-N will be lesser than what it is if $k < 1$.

(2) if $k < 1$ then number of lower levels for which CS1 is employed, are more than what it is, if $k > 1$.
For given values of parameters e and m, the following are different ranges of k, showing various user requirements.

A. For very high specificity and low certainity:

$2.0 < k <= 4.0$

B. For high specificity and moderate certainity:

$1.0 < k <= 2.0$

C. For moderate specificity and moderate certainity:

$k = 1.0$

D. For moderate specificity and high certainity:

$0.5 <= k < 1.0$

E. For low specificity and very high certainity:

$0.25 <= k < 0.5$

fig.4 to fig.6 are for different values of parameter k (0.5, 1.0 and 2.0) and fixed value of parameter m (0.5).

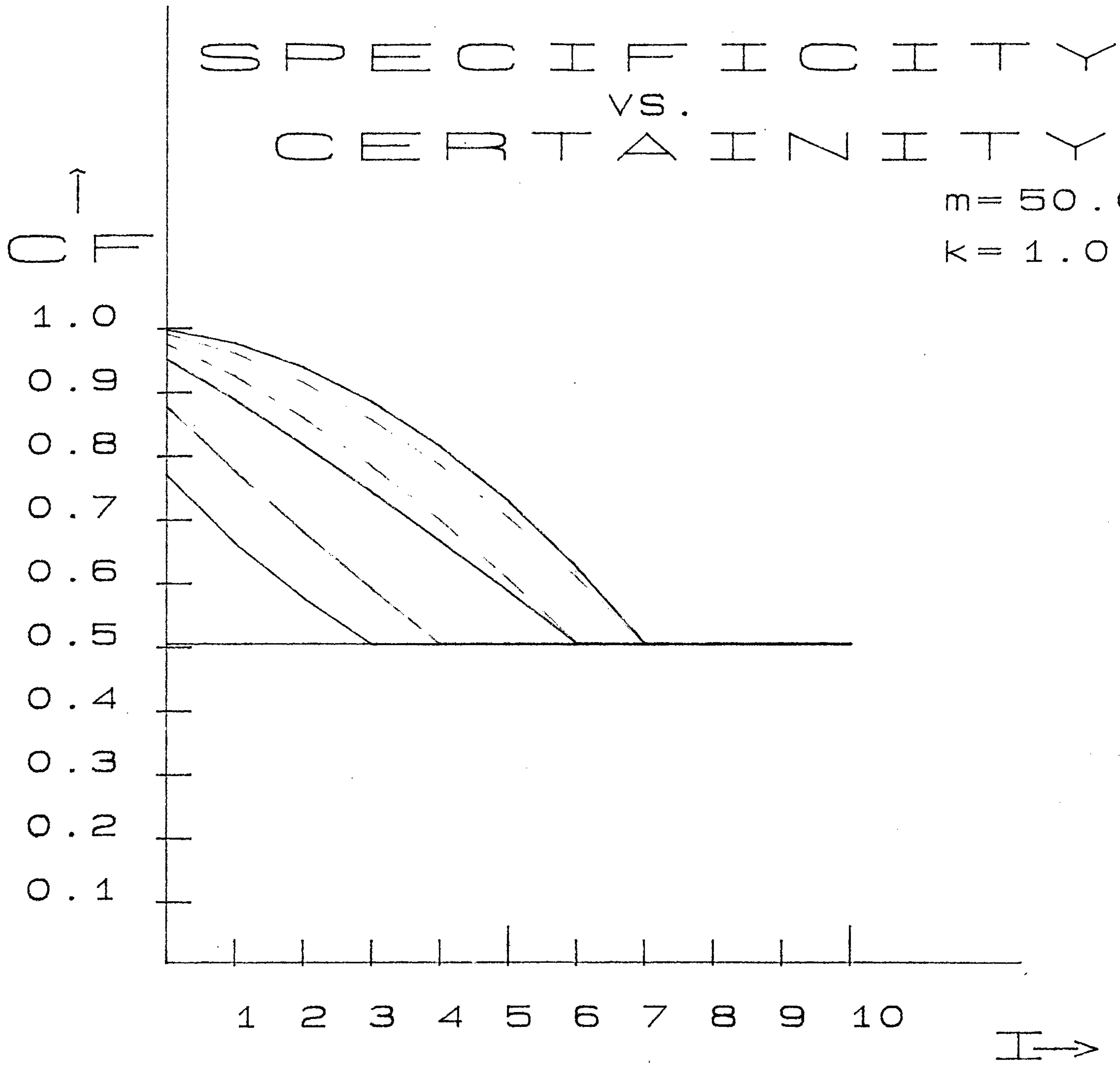
SPECIFICITY (I)
VS.
CERTAINITY (CF)
m= 50.0
k= 1.0
CF
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
1
2
3
4
5
6
7
8
9
10
I

fig..

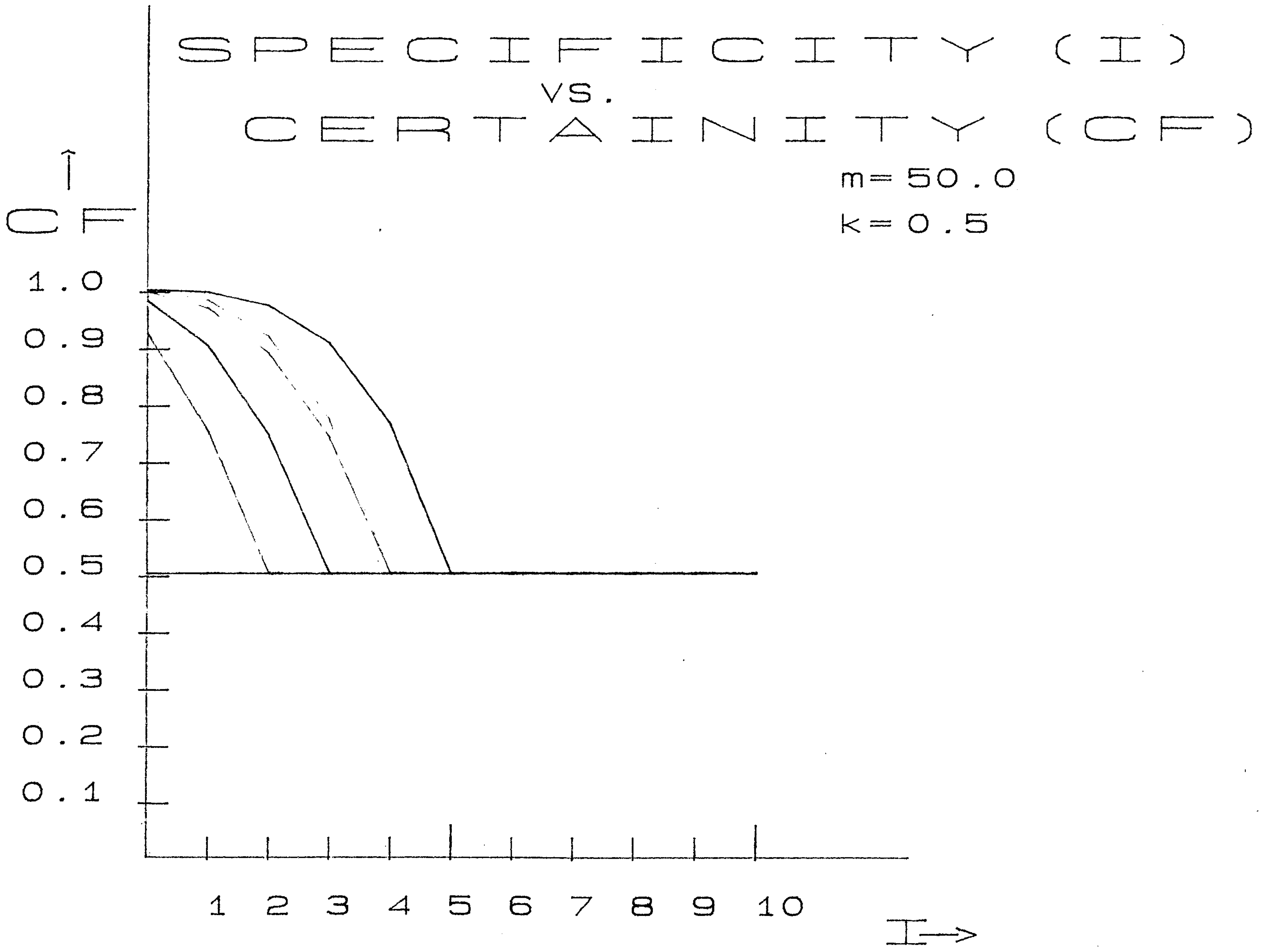
SPECIFICITY (I)
vs.
CERTAINITY (CF)
m= 50.0
k= 0.5
CF
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
1
2
3
4
5
6
7
8
9
10
I→

fig. :

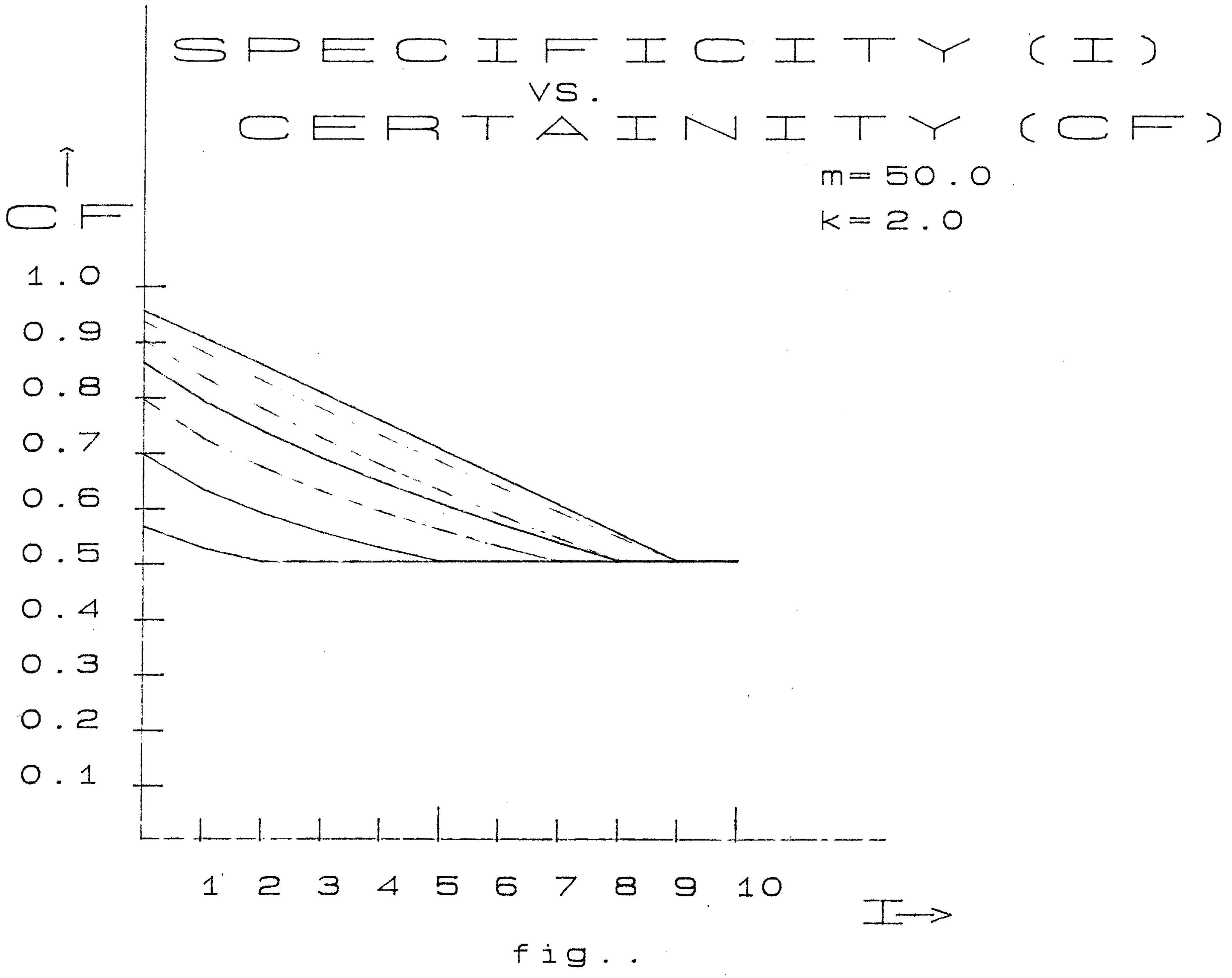
SPECIFICITY (I)
VS.
CERTAINITY (CF)
m= 50.0
k= 2.0
CF
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
1
2
3
4
5
6
7
8
9
10
I

fig..

CHAPTER 5

IMPLEMENTATION

There are two types of knowledge representation items: in a declarative knowledge item, there is nothing to say how it should be used, a procedural item, in contrast, contains within itself explicit information on this point. The former has the nature of an item of data to be used by a program, whilst the latter is the program itself. A declarative item cannot stand alone but must be comlemented by an interpreting procedure; thus a system cannot be fully declarative but it can be fully procedural.

Our implemented intelligent systems employes a procedural knowledge item. This procedural item is based on the HCPRs system of knowledge representation and control informations are relegated to its procedural parts.

The following are the steps of a general scheme employed to implement a HCPR in the practical knowledge-bases.

Step1: A rule is defined by giving it a particular name and a list of control parameters. These control parameters describe the current state of the reasoning process completly. This control information is passed from one rule to another as control is altered between them.

Step2: A variable cfm called minimum certainity factor is initialised to CF of parent rule decision and for the root rule it is 100 (by default).

Step3: A list of exceptions to the rule with their likelihoods and 0-level strength of implication are given.

Step4: It checks the various flags and the ranges of different control parameters.

Step5: Premise part of the rule is defined and value of cfm is updated.

Step6: Employ a control scheme to handle the exception conditions and output (or return) the results of knowledge processing.

Step7: In the last, a set of more specific rules is defined and in forward chaining, control is passed to one of these rule. But, in backward chaining this specificity information is completly neglected.

Using this scheme to represent a HCPR, two knowledge-bases are designed and a rule from the knowledge-base of biological classification is given below:

```
;*************************************************************
;*  This  rule  is for the general concept of living organism ***
;*************************************************************

;.................           STEP1          ....................;
(defun x_is_living_organism(x
  &optional(chaining 'forward)(m 50)(e 1)(k 1)(i 0)
 (n (spe_n 0 m e k)) (y 100)) ;list of optional
;parameters gives control schemes and system's current state.
;.................           STEP2         .....................;
(setq name x  cfm y) ;initialised cfm to CF of parent rule
decision.
;.................           STEP3         .....................;
(let ((s0_impl 96) (elist`((is_it_always_undergoing_
change_in_their_substances_called_metabolism 2 ,name)
(does_it_arise_from_other_organism_of_same_or_
related_species 2 ,name))) ;elist is a list of exceptions &
their likelihood.
;..................         STEP4         ......................;
(and (init_cond_p y m (* e 4) i n) ;check range error.
;..................         STEP5         ......................;
(minip m cfm (posses_p x 'characterstic_shape_&_bound_
by_regular_curved_surface)) ;predicates
(minip m cfm (show_p x 'growth_largely_by_assimlation))
;.................           STEP6          ......................;
(setq c_fac (control_P chaining s0_impl elist cfm x m
 e k i n '| living organism.|)) ;a control scheme is employed.
;.................           STEP7          ......................;
(if (equal chaining 'backward)
        (setq ya c_fac)
  (or(x_is_animalia x chaining m e k (1+ i) n ya1)
     (x_is_plantae x chaining m e k (1+ i) n ya1)
  t))))) ;control is transferred to more specific rules.
```

Another knowledge-base, which gives answers to some daily life queries is designed using a similar knowledge representation scheme. A rule from this knowledge-base is given below:

```
;**************************************************************
;this is a rule for answering a query of the type "is_x_outdoor"*
;**************************************************************

;..................           STEP1          ...................;
(defun is_x_outdoor(x &optional(chaining 'forward)(m 50)(e 1)
(k 1)(i 0)(n(spe_n 1 m e k))(y(what_x_is_doing x 'backward))
;rule name and control parameters.
;.................           STEP2          ...................;
 (setq cfm y) ;initialised the minimum certainity factor cfm.
;.................          STEP3          ....................;
 (let ((elist `((is_bad_weather 4)(is_riots_in_the_city 3)
  (is_final_exams_in_progress 3 ,name)(is_he_ill 3 ,name))))
 ;a list of exceptions to the rule.
;.................          STEP4          ....................;
  (and (init_cond_p y m (* e 4) i n) ;checks ranges & flags.
;................           STEP5          ....................;
         (setq cfm (min(day_p 'sunday) cfm)) ;premise part.
;................           STEP6          ....................;
   (if (equal chaining 'backward) ;return CF of decision.
             (exception_handler 85 elist 100 e 1 1)
         (and ( > (exception_handler 85 elist m e i n k) m)
                     (writep x '| is outdoor . | i)
;output a decision with calculated certainity factor.
;................           STEP7          ....................;
                      (setq ya1 ya m1 m e1 e)
;control is transferred to more specific set of rules.
(or(is_x_playing_outdoor x chaining m1 e1 k (1+ i) n ya1)
(is_x_entertainning_outdoor x chaining m1 e1 k (1+ i) n ya1)
(is_x_working_outdoor x chaining m1 e1 k (1+ i) n ya1)t))))))
```

For these implemented intelligent systems, a certainity factor in the range of 0 to 100 (instead of 0 to 1) is used and it denotes the percentage certainity factor, for simplicity we would call it CF.

In the rule of living organism a macro "control_p" is used, it simply returns the value of certainity factor of the decision if chaining is backward, otherwise, it will output the decision with its truth value.
Appendix-A includes macro "control_p" alongwith some other important functions. Appendix-B and Appendix-C include example sessions with these knowledge-bases, for different control parameters.

CHAPTER 6

OTHER OBSERVATIONS: HCPRs SYSTEM

The power of a method of representation can be judged by

- its ability to express complex situations precisely.
- its clarity of representation.
- its efficient utilisation of available memory.
- its ability to deal with imprecise arguments.
- its ability to reason with insufficient data.
- its ability to conclude both the positive and negative inferences.
- its ability to facilitate parallel processing.
- its ability to remember past experiences.

- its ability to improve performance as time passes, or its ease to learning, and in the last
- it should be an ideal representation scheme.

Most of these features of the HCPRs system of knowledge representation have been discussed in previous chapters. In this chapter a set of inference rules, a parallel processing scheme and some learning schemes for HCPRs system of knowledge representation are described as follows:

6.1 INFERENCE RULES

Consider a HCPR without censor conditions or simply a hierarchical production rule (HPR) and a rule-tree R1, of HPRs:

```
> R1:

      #  A ==> B $ (D1 xor D2)

      #  B and A1 ==> D1 $ (E1 xor E2)

      #  B and A2 ==> D2 $ (F1 xor F2)

      #  D1 and A11 ==> E1 $ (UNKW1)

      #  D1 and A12 ==> E2 $ (UNKW2)
```

D2 and A21 ==> F1 $ (UNKW3)

D2 and A22 ==> F2 $ (UNKW4)

Where, UNKW1,UNKW2,.. represent unknown specificity informations to the rules. We are assuming that the specificity informations to the rules of decisions "B", "D1" and "D2" are complete. Otherwise, the incomplete specificity information to the rule of decision "D1" would be written as (E1 xor E2 xor UNKNOWN). A sample of inference rules applicable to a HPRs system of knowledge representation is given below:

```
              |
R1            |
              |>  B:true
A:true        |
              |


              |
R1            |
              |>  B:false
A:false       |
              |


              |
R1            |
              |>  D1:false
B:false       |>  D2:false
              |>  A:false
              |
          |
R1        |
          |>   D1:true
B:true    |
          |>   D2:false
A1:true   |
          |


          |
R1        |
          |>   F1:false
D2:false  |
          |>   F2:false
          |


          |
R1        |
          |>   E1:false
D1:true   |
          |>   E2:true
A12:true  |
          |


          |
R1        |
          |>   UNKW2:true
E2:true   |>   E1:false
          |


          |
R1        |
          |>   UNKW4:false
F2:false  |
          |
                |
R1              |
                |>  A:true
B:true          |
                |
```

```
                |
R1              |
                |>  B:true
D2:true         |>  A2:true
                |>  D1:false
                |
```

```
                |
R1              |
                |>  A21:true
                |>  D2:true
F1:true         |>  F2:false
                |>  UNKW3:true
                |
```

```
                |
R1              |
                |
F2:false        |> A22:false
                |
D2:true         |
                |
```

As an example consider the following rule-tree "Rex", representing the general concept of excitement:

```
                    EXCITEMENT
                    /         \
                   /           \
                  /             \
                 /               \
                /                 \
        Distress                   Delight
        /   |   \                 /   |   \
       /    |    \               /    |    \
   Fear   Shame  Anger   Affection   Joy    Elation
```

fig.- 1

The rule-tree "Rex" in fig.1, may be given as follows:

> Rex:

```
    #  State of agitation  ==> Excitement $
                                        (Distress  xor Delight)

    #  Excitement and extreme pain  ==> Distress $
                                     (Fear xor Shame xor Anger)

    #  Excitement and great pleasure  ==> Delight $
                                (Affection xor Joy xor Elation)

    #  Distress and danger               ==>   Fear

    #  Distress and guilt feeling        ==>   Shame

    #  Distress and real/fancied injury ==>   anger
```

#	Delight and kindness/love	==>	Affection
#	Delight and gladness	==>	Joy
#	Delight and pride from success	==>	Elation

This rule-tree may be used to find a proper word, which is required to describe the feeling of a person. On noticing that the person is in the state of agitation, the reasoning system with rule-tree "Re" will infer that "he is excited". After noticing the fact that "he is having great pleasure", it will infer that "he is delighted". Similarly, on finding further that "he is showing great gladness", it will infer more specifically that "he is in joy." And, on taking into account the mutually exclusive property between sibling rules and the fact that "he is in joy ", it may infer the following statements (negative):

1. he is not feeling shame.

2. he is not fearful.

3. he is not angry.

4. he is not elated.

5. he is not having affection.

6. he is not distressed.

6.2 LEARNING

The rule-trees in a HCPRs system have the capability of continuous growth with time. A rule-tree will become stronger (strength of implication) and richer in knowledge as time passes. Like exceptions, specificity informations may be incomplete or even absent in a HCPR depending on, how much a system has learnt. Following are some of the possible learning schemes suitable for the HCPRs system.

6.2.1 Remembering most likely Lines of Action

Specificity informations may be resequenced according to how frequently a specific rule has been applied in the past or in the order of decreasing importance. Such that, information of rule, which is used most frequently (or recently) should come first in the specificity information part of the rule. A HCPRs based reasoning system should apply rules according to their order in the specificity information, rather than the order in which they are stored in the rule base. Similarly, exception conditions may be stored in the order of their cost-factor and likelihoods.

6.2.2 Learning by Refining Beliefs

Strength of implications (0-level) and liklihoods of various exceptions should be updated according to the past experience of the system. This could be performed by using the past-data of each HCPR, where the past-data may include information of the type:

- number of times a HCPR has been employed successfully in the past,
- number of times a particular exception has blocked the rule, etc.

6.2.3 Learning by Concept Formation

To understand learning by concept formation consider an example: The children's concepts often show a crude generality which has to be overcome by taking, note of differences. A little girl on seeing a squirrel called it a "funny kitty" . She was generalising by assimilating the new to the old, but as she noticed that the new kitty was funny, she was ready to draw a new name and differentiate a new concept.

6.2.3.1 Horizontal Growth - If a new rule to be added to a rule-tree, does not increase the maximum level of specificity, then its inclusion in the rule-tree will be called its horizontal growth (fig.2 and fig.3).

```
         A1                A1
        /  \              /  \
       /    \            /    \
     A2      A3   ==>  A2      A3
     |                /  \
     |               /    \
     A4             A4    A5
```

Fig.- 2

```
         A1                A1
        /  \              /  \
       /    \            /    \
     A2      A3   ==>  A2      A3
     |                 |       |
     |                 |       |
     A4               A4       A5
```

Fig.- 3

Horizontal - Growth

6.2.3.2 Vertical Growth - If a new rule to be added to a rule-tree, increases the maximum level of specificity by one, then its inclusion in the rule-tree will be called its vertical growth (fig.4).

```
          A1                A1
         /  \              /  \
        /    \            /    \
      A2      A3   ==>  A2      A3
     /  \     |        /  \     |
    /    \    |       /    \    |
  A4     A5  A6     A4     A5   A6
                           |
                           |
                           A7
```

Fig.- 4

Vertical - Growth

6.2.4 Learning by Fusion

Consider two rule-trees R1 and R2 (fig.5), which represent two unrelated concepts "Aa1" and "Ab1". But, after some time it is observed that these two concepts are related, so they may be combined in one single general concept by introducing a new rule C1, with rule-trees R1 and R2 its subtrees.

```
     Aa1                 Ab1               C1
    /   \       +       /   \      ==>    /  \
   /     \             /     \           /    \
 Aa2      Aa3        Ab2     Ab3       Aa1     Ab1
                                      /  \    /  \
    -R1-                -R2-         /    \  /    \
                                   Aa2    Aa3 Ab2   Ab3

                                           -R3-
```

fig.- 5

A rule-tree R2, with an independent concept "Aa1" initially, may be a sub concept of another rule-tree R1. So, it would become a subtree of rule-tree R1 after suitable modification (fig.6).

```
        Aa1          Ab1          Aa1
       /   \        /   \        /   \
      /     \   +  /     \  ==> /     \
    Aa2    Aa3    Ab2    Ab3   Aa2     Aa3
     |                        /   \
     |                       /     \
    Aa4            -R2-    Aa4     Ab1
                                  /   \
        -R1-                     /     \
                               Ab2     Ab3

                                   -R1-
               fig.- 6
```

6.2.5 Learning by Fission

This process is employed to simplify a complex rule by breaking it into two or more simpler rules, related to each other in hierarchy (fig.7). This process will be called vertical fission.

```
          A1                  A1
         /   \               /  \
        /     \             /    \
   A2          A3   ==>    B1     A3
  /  \         |           |      |
 /    \        |           |      |
A4     A5      A6          B2     A6
                          /  \
      -R1-               /    \
                        A4     A5

                           -R1-
```

fig.- 7

Similarly as the number of sibling rules crosses some critical value they may split under two or more intermediate parent rules (fig.8). This process will be called horizontal fission.

```
       A1                          A1
      /  \                        /  \
     /    \       ==>            /    \
    A2     A3                   A2     A3
  / | \                        /  \
 /  |  \                      /    \
A4  A5  A6                  B1      B2
                           /  \     |
   -R1-                   /    \    |
                        A4      A6  A5

                                   -R1-
```

fig.- 8

where, B1 and B2 are two intermediate parent rules.

6.3 PARALLEL PROCESSING

The HCPRs system of knowledge representation supports parellel processing. At each level of specificity there would be a set of rules (child rules), which is the most relevant to the current state of the system. A HCPRs system of knowledge representation provides an efficient mechanism to get information about this set of rules and how the rules are related to each other. So, according to their mutual relationships they may be assigned to separate processors, i.e., all the mutually exclusive rules might be assigned to separate processors and processor finising first with successful and acceptable results might abort the jobs of remaining processors and these processors may be reassigned to derive more specific inferences. Here, successful and acceptable results mean answers with certainity factor greater than m. Also, if it has to choose between more than one answers (not mutually exclusive) then the answer with the largest CF would be selected.

CHAPTER 7

CONCLUSION

Our aim is to produce systems having tens of thoushands of rules or even more. These systems will have several levels of knowledge, the higher levels making possible a more intelligent use of the lower: if a program is asked to find out "what book John is reading" and it is known to him that he is sleeping then it should not search to find this.

Both human and computer must be able to react promptly to new information, and they must be able to change or repair their knowledge when new information produces contradictions or when initial assumptions are withdrawn.

For lack of any better way, information scientists today continue to improve the knowledge bases of their programs by hand. The small size of existing programs makes this acceptable, but what will happen when they get to the size of a million rules or concepts ? Programs on this scale will themselves have to learn from experience and to improve themselves by using simple rules provided by human experts for assessing their performance.

The less sophisticated programs simply read the data at the start of the session, perform the reasoning operations on these and give the results without taking any advantage of the interactive possibilities offered by the computer. Also sometimes all the data relevant to the problem is not available at the time when solution is required. The consequence of the data being incomplete is simply that the conclusions are less certain, or less good in some sense - with the possibility of being wrong in some cases. Also an answer which is more certain but somewhat general is sometimes acceptable. Thus, our programs should be capable of providing some solutions, even somewhat general or less certain.
It is also possible that some data which is unrelated to the problem (or noise information) is provided. Our programs should be able to discard these task irrelevant informations efficiently and should concentrate on the main line of reasoning.

In the present work, we have suggested a HCPRs system of knowledge representation, which has the ability to incorporate into the knowledge structure additional information that can be used to focus the attention of the inference mechanism in the most promising directions.
We have shown that a HCPRs system enables the trade-offs between the precision of decisions and the efforts needed to derive those

decisions.

The control planning problem is one of the important topic, which has been described using a general control scheme (GCS). The GCS describes how a reasoning mechanism for the HCPRs system, can make controllable trade-offs between the certainity and the specificity.
It has been shown that a HCPRs system supports various learning schemes and how it can improve its reasoning power using past experience.
Also, a parallel processing scheme for a HCPRs system has been described.

We need to find some more suitable operators
using which various mutual relationships between different rules, conditions and facts can be made explicit, i.e., some suitable operators to give relationships of the type Has_parts, Has_elements, Has_properties,... and Has_constraints are required.
Like specificity operator "$", we think, a generality operator "G%" should be employed, such that, our representation of a rule-tree becomes:

```
A0 ==> B $ (C xor D)
A1 ==> C $ (C1 xor ..) G% B
A2 ==> D $ (D2 xor..) G% B
A11 ==> C1 $ (....) G% C
::::::::::
```

instead of the following rule-tree representation:

A0 ==> B $ (C xor D)

B and A1 ==> C $ (C1 xor ..)

B and A2 ==> D $ (D2 xor..)

:::::::::::::

Operator "G%" would make the backward chaining equally efficient as the forward chaining. In backward chaining a reasoning process should neglect the information associated with the operator "$" and in forward chaining information associated with operator "G%" should be neglected.

We have implemented HCPRs system using procedural representation of knowledge, but the power of the HCPRs system is in its declarative representation of knowledge. Implementation of the HCPRs system with declarative knowledge representation, would certainly require an interpreter capable of handling the various features of the HCPRs system of knowledge representation.

APPENDIX A

LISTING OF IMPORTANT FUNCTIONS DEFINED IN LISP

```
;*************************************************************
;**   a macro to control the forward and backward chaining.  ****
;*************************************************************

(defmacro control_p(chaining s0_impl elist cfm x m e k i n s1)
 `(if (equal ,chaining 'backward) ;backward chaining
     (exception_handler ,s0_impl ,elist)
  (if (> (setq ya (* ,s0_impl ,cfm .01)) ,m) ;forward chaining
      (and (writep ,x '| belongs to| ,s1 ,i) ;output decision
 (if ( > (exception_handler ,s0_impl ,elist, m, e ,k ,i ,n) m)
        (and (print `(is true with cf = ,ya)) ;output CF
            (print `(and required cf was ,cf)))
        (and (print `(is false with cf = 100)) (= 1 2)))
               (setq ya1 ya)))))


;*************************************************************
;***        a function for general exception handler      *******
;*************************************************************

(defun exception_handler(y elist m e i &optional (n  nm) (k 1))
  (setq cf (- 100.0 (* (- 100 m) (expt(/ (+ i 1) n) (/ e k)))))
       (setq c y prede elist)
       (do (   )
           ((or (< cfm m)          ;IF premise part is false,
           (equal (* c cfm .01) 0.0) ;  an exception is true,
          (> (/ (* c cfm) 100.0) cf);  required CF is achieved
             (null prede)) (setq ya (* c cfm .01)) ;or no more
;exception remains THEN return calculated CF of decision.
       (handle (car prede)) ;ELSE check next exception to rule
          (setq prede (cdr prede))))
```

```
;*****************************************************************
;***     this function modify the strength of implication     ****
;*****************************************************************

(defun handle(x)
   (setq excep (exceptions  x)) ;get the truth value of
;exception conditon x.
       (cond ((equal excep 'y) (setq c 0));if true then
;strenght of implication is 0.0
              ((equal excep 'n) (setq c (+ c (cadr x))))
;if false then strenght of implication is increased.
               (t))) ; if unknown it remains same.



;*****************************************************************
;**      This function find the truth value of an exception    ***
;*****************************************************************

(defun exceptions(x)
 (cond ((caar(my_member  x exception_list)))
;if truth value of exception is known then return it.
       (t(and (print`(,(car x) ? write Y/N.))
;otherwise request from the user.
         (setq ans (read))
         (inter_stop ans)
        (cond ((or (eq ans 'n)(eq ans 'y))
(and(setq exception_list(cons(list ans x) exception_list))
;update exception list.
     (caar exception_list)))
     (t 'q))))))



;****;**********************************************************
;***      this function calculate the maximum level of     *****
;**      specificity to be tried in given conditions     *******
;***************************************************************

(defmacro spe_n (x m e k)
 `(ceiling (* (- ,nm ,x ) (expt (* (- 100.0 ,m) 0.01)
                               (/ 1.0 (* ,k ,e))))))
```

```
;*************************************************************
;**     this function reset various flags and database.  *****
;*************************************************************

(defun reset()
     (setq dynamic_database1 '() dynamic_database2 '()
      exception_list '() get_list '()   nm 12  flag nil))


;*************************************************************
;**    this function checks the range of various parameters  ***
;*************************************************************

(defun init_cond_p(a &optional b c d e)
      (if (and (not flag)) (and (cf_range_p a)
                                (cf_range_p b)
                                (cf_range_p c)
                                (cf_range_p d)
                                (cf_range_p e))
                          (= 1 2)))

(defun cf_range_p(x)
       (if (stringp x) (= 1 2)
       ((<= 0 x 100)))


;*************************************************************
;*****            this function update cfm           *********
;*************************************************************

(defun minip( x y z)
        (cond ((>= z x) (setq  cfm (min y z)))
              ('())))


;*************************************************************
;*   this function is invoked when resources are expired   ***
;*************************************************************

(defun constraints(m)
       (print `(do you want to proceed further? write y/n.))
       (if (equal (read) 'y)
           (setq cf m)
           (setq c 0 flag 'true)))
```

APPENDIX B

EXAMPLE SESSIONS: KNOWLEDGE-BASE 1

```
;*****************************************************************
;***          EXAMPLE SESSIONS (KB1) START FROM HERE       ********
;*****************************************************************
Lisp>
(reset)
NIL

Lisp> (what_x_is_doing 'k.lal 'forward 40 2 .75)
(GIVE THE NAME OF CITY OF WHICH K.LAL IS RESIDENT.)
delhi
(GIVE CF (0 TO 100) THAT K.LAL IS RESIDENT OF DELHI CITY IS
TRUE.) 100
(HAS_HE_LONG_VACATION ? WRITE Y/N.) n
(IS_HIS_CLOSE_RELATIVE_SERIOUSLY_ILL_IN_ANOTHER_CITY ? WRITE
Y/N.) n
(IS_HIS_CLOSE_RELATIVE/FRIEND_HAS_MARRIGE_FUNCTION_IN_OTHER_CITY
? WRITE Y/N.) n
(IS_HE_A_SALES_PERSON ? WRITE Y/N.) n

SPECIFICITY_LEVEL_IS 0

DECISION_IS K.LAL is in the DELHI
ITS_CF_IS 99.0
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 99.17921
(GIVE CF (0 TO 100) THAT DAY IS SUNDAY IS TRUE.) 100
(IS_BAD_WEATHER ? WRITE Y/N.) n
(IS_RIOTS_IN_THE_CITY ? WRITE Y/N.) n
(IS_FINAL_EXAMS_IN_PROGRESS ? WRITE Y/N.) n
(IS_HE_ILL ? WRITE Y/N.) n

SPECIFICITY_LEVEL_IS 1
DECISION_IS K.LAL is outdoor .

ITS_CF_IS 97.01999
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 94.78831
```

```
(GIVE CF (0 TO 100) THAT IT IS PLAYING TIME IS TRUE.) 10
(GIVE CF (0 TO 100) THAT IT IS ENTERTAINNING TIME IS TRUE.) 100
(DID_HE_GOT_A_MAJOR_ACCIDENT ? WRITE Y/N.) n

SPECIFICITY_LEVEL_IS 2

DECISION_IS K.LAL is entertainning outdoor .
ITS_CF_IS 87.318
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 84.63422
(GIVE CF (0 TO 100) THAT K.LAL LIKES SEA IS TRUE.) 75

SPECIFICITY_LEVEL_IS 3

DECISION_IS K.LAL is at sea side .
ITS_CF_IS 67.5
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 66.90788
(GIVE CF (0 TO 100) THAT K.LAL HAS ENOUGH_SPARE_TIME IS TRUE.) 70

SPECIFICITY_LEVEL_IS 4

DECISION_IS K.LAL is sitted at sea side .
ITS_CF_IS 60.75
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 40.0
T

Lisp> (what_x_is_doing 'k.lal 'forward 60 2.0 2.0)

SPECIFICITY_LEVEL_IS 0

DECISION_IS K.LAL is in the DELHI
ITS_CF_IS 94.0
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 92.0

SPECIFICITY_LEVEL_IS 1

DECISION_IS K.LAL is outdoor .
ITS_CF_IS 86.48
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 84.0

SPECIFICITY_LEVEL_IS 2

DECISION_IS K.LAL is entertainning outdoor .
ITS_CF_IS 77.832
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 76.0
(IS_CITY_NOT_AT_SEA_SIDE ? WRITE Y/N.)
n

SPECIFICITY_LEVEL_IS 3

DECISION_IS K.LAL is at sea side .
```

```
ITS_CF_IS 71.25
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 68.0

SPECIFICITY_LEVEL_IS 4

DECISION_IS K.LAL is sitted at sea side .
ITS_CF_IS 63.0
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 60.0
T

Lisp> (is_x_outdoor 'k.lal 'backward)
84.15

Lisp> (what_x_is_doing 'k.lal 'forward  50 .5 .5)

SPECIFICITY_LEVEL_IS 0

DECISION_IS K.LAL is in the DELHI
ITS_CF_IS 85.0
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 50.0
T

Lisp> (what_x_is_doing 'k.lal 'forward  50 .5 2.)

SPECIFICITY_LEVEL_IS 0

DECISION_IS K.LAL is in the DELHI
ITS_CF_IS 85.0
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 62.00821

SPECIFICITY_LEVEL_IS 1

DECISION_IS K.LAL is outdoor .
ITS_CF_IS 72.25
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 54.8199

SPECIFICITY_LEVEL_IS 2

DECISION_IS K.LAL is entertainning outdoor .
ITS_CF_IS 61.41249
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 50.0
T

Lisp> (what_x_is_doing 'k.lal 'forward  80 .5 2.)

SPECIFICITY_LEVEL_IS 0

DECISION_IS K.LAL is in the DELHI
ITS_CF_IS 85.0
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 83.18207
```

```
SPECIFICITY_LEVEL_IS 1

DECISION_IS K.LAL is outdoor .
ITS_CF_IS 80.75
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 80.0
T

Lisp> (what_x_is_doing 'k.lal 'forward 40 2 2)

SPECIFICITY_LEVEL_IS 0

DECISION_IS K.LAL is in the DELHI
ITS_CF_IS 94.0
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 90.0

SPECIFICITY_LEVEL_IS 1

DECISION_IS K.LAL is outdoor .
ITS_CF_IS 83.66
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 80.0

SPECIFICITY_LEVEL_IS 2

DECISION_IS K.LAL is entertainning outdoor .
ITS_CF_IS 71.111
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 70.0

SPECIFICITY_LEVEL_IS 3

DECISION_IS K.LAL is at sea side .
ITS_CF_IS 63.9999
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 60.0

SPECIFICITY_LEVEL_IS 4

DECISION_IS K.LAL is sitted at sea side .
ITS_CF_IS 57.59991
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 50.0

SPECIFICITY_LEVEL_IS 5

DECISION_IS K.LAL is sitted at sea side alongwith his wife.
ITS_CF_IS 51.83992
CF_REQUIRED_BY_CONTROL_SCHEME_WAS 40.0
T
Lisp>
(exit)
```

APPENDIX C

EXAMPLE SESSIONS: KNOWLEDGE-BASE 2

```
;*****************************************************************
;****        EXAMPLE SESSION (KB2) START FROM HERE         ******
;*****************************************************************

Lisp>
(reset)
NIL

Lisp> (x_is_living_organism 'X 'forward 80 2.0 1.0)
(GIVE CF (0 TO 100) THAT X POSSESS
CHARACTERSTIC_SHAPE_&_BOUND_BY_REGULAR_CURVED_SURFACE IS TRUE.)
100
(GIVE CF (0 TO 100) THAT X SHOWS GROWTH_LARGELY_BY_ASSIMLATION IS
TRUE.) 100

SPECIFICITY_LEVEL_IS 0

DECISION_IS X belongs to living organism.
(IS_IT_ALWAYS_UNDERGOING_CHANGE_IN_THEIR_SUBSTANCES_CALLED_
METABOLISM ?)  (WRITE T/F ?.) t
(DOES_IT_ARISE_FROM_OTHER_ORGANISM_OF_SAME_OR_RELATED_SPECIES ?)
(WRITE T/F ?.) t
(IT IS TRUE WITH CF = 100.0)
(AND CF REQUIRED BY CONTROL SCHEME WAS 99.44444)
(GIVE CF (0 TO 100) THAT X POSSESS CHLOROPHYL IS TRUE.) 10

SPECIFICITY_LEVEL_IS 1

DECISION_IS X belongs to Kingdom Animalia.
(DOES_IT_TAKE_ORGANIC_FOOD_PRODUCECED_BY_OTHER_ORGANISMS ?)
(WRITE T/F ?.) t
(IT IS TRUE WITH CF = 90.0)
(AND CF REQUIRED BY CONTROL SCHEME WAS 97.77777)
(GIVE CF (0 TO 100) THAT X HAS MULTICELLULAR_BODY IS TRUE.) 100
(GIVE CF (0 TO 100) THAT X SHOWS
ORGANISATION_AT_TISSUE_LEVEL_OR_ORGAN_LEVEL IS TRUE.) 100

SPECIFICITY_LEVEL_IS 2

DECISION_IS X belongs to Subkingdom Metazoa.
(IT IS TRUE WITH CF = 89.09999)
(AND CF REQUIRED BY CONTROL SCHEME WAS 95.0)
(GIVE CF (0 TO 100) THAT X HAS
UNIQUE_SUPPORTING_STRUCTURE_THE_NOTOCHORD IS TRUE.) 100
(GIVE CF (0 TO 100) THAT X HAS SEGMENTED_BODY IS TRUE.) 100
(GIVE CF (0 TO 100) THAT X HAS BILATERAL_SYMMETRY IS TRUE.) 100

SPECIFICITY_LEVEL_IS 3

DECISION_IS X belongs to Phylum Chordata.
(DOES_IT_HAS_CLOSED_BLOOD_SYSTEM ?)
(WRITE T/F ?.) t
(IT IS TRUE WITH CF = 89.09999)
(AND CF REQUIRED BY CONTROL SCHEME WAS 91.11111)
(GIVE CF (0 TO 100) THAT X HAS BRAIN_OR_SKULL_OR_HEAD IS TRUE.)
100
(GIVE CF (0 TO 100) THAT X HAS ENDOSKELETON_OF_BONE_OR_CARTILAGE
IS TRUE.) 100

SPECIFICITY_LEVEL_IS 4

DECISION_IS X belongs to Sub-Phylum Craniata.
(IT IS TRUE WITH CF = 88.20899)
(AND CF REQUIRED BY CONTROL SCHEME WAS 86.11111)
(GIVE CF (0 TO 100) THAT X HAS JAWS IS TRUE.) 100

SPECIFICITY_LEVEL_IS 5

DECISION_IS X belongs to Super-Class Gnathostomata.
(IT IS TRUE WITH CF = 84.68063)
(AND CF REQUIRED BY CONTROL SCHEME WAS 80.0)
(GIVE CF (0 TO 100) THAT X HAS BODY_COVERED_WITH_HAIR IS TRUE.)
100
(GIVE CF (0 TO 100) THAT X POSSESS MAMMARY_GLANDS IS TRUE.) 100

SPECIFICITY_LEVEL_IS 6

DECISION_IS X belongs to Class Mammalia.
(DO YOU WANT TO PROCEED FURTHER? WRITE Y/N.) y
(IT IS TRUE WITH CF = 84.68063)
(AND CF REQUIRED BY CONTROL SCHEME WAS 80)
(GIVE CF (0 TO 100) THAT X POSSESS POUCH IS TRUE.) 1
(GIVE CF (0 TO 100) THAT X HAS
MATURE_NEW_BORN_WITH_COVERING_OF_HAIRS_&_ALL_SENSE_FUNCTIONING IS
TRUE.) 100
```

```
SPECIFICITY_LEVEL_IS 7

DECISION_IS X belongs to Sub-Class Eutheria.
(DO YOU WANT TO PROCEED FURTHER? WRITE Y/N.) y
(IT IS TRUE WITH CF = 82.98702)
(AND CF REQUIRED BY CONTROL SCHEME WAS 80)
(GIVE CF (0 TO 100) THAT X POSSESS
CLOSED_BONY_RINGS_SURROUNDED_TO_EYE_SOCKET IS TRUE.) 100
(GIVE CF (0 TO 100) THAT X HAS
1ST_FINGER_&_AT_LEAST_ONE_PAIR_OF_DIGIT_OPPOSABLE_&_ABLE_TO_GRASP
IS TRUE.) 100

SPECIFICITY_LEVEL_IS 8

DECISION_IS X belongs to Order Primates.
(DO YOU WANT TO PROCEED FURTHER? WRITE Y/N.) y
(IT IS TRUE WITH CF = 81.32727)
(AND CF REQUIRED BY CONTROL SCHEME WAS 80)
(GIVE CF (0 TO 100) THAT NAKED_MOIST_RHINARIUM IS PRESENT IS
TRUE.) 0
T

Lisp> (x_is_living_organism 'Y 'forward 50)
(GIVE CF (0 TO 100) THAT Y POSSESS
CHARACTERSTIC_SHAPE_&_BOUND_BY_REGULAR_CURVED_SURFACE IS TRUE.)
100
(GIVE CF (0 TO 100) THAT Y SHOWS GROWTH_LARGELY_BY_ASSIMLATION IS
TRUE.) 100

SPECIFICITY_LEVEL_IS 0

DECISION_IS Y belongs to living organism.
(IT IS TRUE WITH CF = 96.0)
(AND CF REQUIRED BY CONTROL SCHEME WAS 91.66666)
(GIVE CF (0 TO 100) THAT Y POSSESS CHLOROPHYL IS TRUE.) 10

SPECIFICITY_LEVEL_IS 1

DECISION_IS Y belongs to Kingdom Animalia.
(IT IS TRUE WITH CF = 86.4)
(AND CF REQUIRED BY CONTROL SCHEME WAS 83.33333)
(GIVE CF (0 TO 100) THAT Y HAS MULTICELLULAR_BODY IS TRUE.) 100
(GIVE CF (0 TO 100) THAT Y SHOWS
ORGANISATION_AT_TISSUE_LEVEL_OR_ORGAN_LEVEL IS TRUE.) 100

SPECIFICITY_LEVEL_IS 2

DECISION_IS Y belongs to Subkingdom Metazoa.
(IT IS TRUE WITH CF = 85.536)
(AND CF REQUIRED BY CONTROL SCHEME WAS 75.0)
```

```
(GIVE CF (0 TO 100) THAT Y HAS
UNIQUE_SUPPORTING_STRUCTURE_THE_NOTOCHORD IS TRUE.)
100
(GIVE CF (0 TO 100) THAT Y HAS SEGMENTED_BODY IS TRUE.) 100
(GIVE CF (0 TO 100) THAT Y HAS BILATERAL_SYMMETRY IS TRUE.) 100

SPECIFICITY_LEVEL_IS 3

DECISION_IS Y belongs to Phylum Chordata.
(IT IS TRUE WITH CF = 83.82528)
(AND CF REQUIRED BY CONTROL SCHEME WAS 66.66667)
(GIVE CF (0 TO 100) THAT Y HAS BRAIN_OR_SKULL_OR_HEAD IS TRUE.)
100
(GIVE CF (0 TO 100) THAT Y HAS ENDOSKELETON_OF_BONE_OR_CARTILAGE
IS TRUE.) 100

SPECIFICITY_LEVEL_IS 4

DECISION_IS Y belongs to Sub-Phylum Craniata.
(IT IS TRUE WITH CF = 82.98703)
(AND CF REQUIRED BY CONTROL SCHEME WAS 58.33333)
(GIVE CF (0 TO 100) THAT Y HAS JAWS IS TRUE.) 100

SPECIFICITY_LEVEL_IS 5

DECISION_IS Y belongs to Super-Class Gnathostomata.
(IT IS TRUE WITH CF = 79.66754)
(AND CF REQUIRED BY CONTROL SCHEME WAS 50.0)
(GIVE CF (0 TO 100) THAT Y HAS BODY_COVERED_WITH_HAIR IS TRUE.)
100
(GIVE CF (0 TO 100) THAT Y POSSESS MAMMARY_GLANDS IS TRUE.) 90

SPECIFICITY_LEVEL_IS 6

DECISION_IS Y belongs to Class Mammalia.
(DO YOU WANT TO PROCEED FURTHER? WRITE Y/N.) y
(IT IS TRUE WITH CF = 79.66754)
(AND CF REQUIRED BY CONTROL SCHEME WAS 50)
(GIVE CF (0 TO 100) THAT Y POSSESS POUCH IS TRUE.) 20
(GIVE CF (0 TO 100) THAT Y HAS
MATURE_NEW_BORN_WITH_COVERING_OF_HAIRS_&_ALL_SENSE_FUNCTIONING IS
TRUE.) 0
(GIVE CF (0 TO 100) THAT Y LAY SHELLED_EGGS IS TRUE.) 100
```

```
SPECIFICITY_LEVEL_IS 7

DECISION_IS Y belongs to Sub-Class Prototheria.
(DO YOU WANT TO PROCEED FURTHER? WRITE Y/N.) y
(IT IS TRUE WITH CF = 78.07419)
(AND CF REQUIRED BY CONTROL SCHEME WAS 50)
(GIVE CF (0 TO 100) THAT Y IS
PLUMP_STOUT_SHORT_LEGGED_&_SHORT_TAILED IS TRUE.) 100
(GIVE CF (0 TO 100) THAT Y HAS THICK_FUR_OR_HAIR_&_SPINES IS
TRUE.) 90

SPECIFICITY_LEVEL_IS 8

DECISION_IS Y belongs to Order Monotremata.
(DO YOU WANT TO PROCEED FURTHER? WRITE Y/N.) y
(IT IS TRUE WITH CF = 78.07419)
(AND CF REQUIRED BY CONTROL SCHEME WAS 50)
(GIVE CF (0 TO 100) THAT Y HAS WT._BETWEEN_2.5_TO_6_KG. IS TRUE.)
100
(GIVE CF (0 TO 100) THAT Y HAS
BEAK_LIKE_SNOUT_OF_ROUND_CROSECTION IS TRUE.) 80

SPECIFICITY_LEVEL_IS 9

DECISION_IS Y belongs to Family Tachyglossidre.
(DO YOU WANT TO PROCEED FURTHER? WRITE Y/N.) y
(IT IS TRUE WITH CF = 77.29344)
(AND CF REQUIRED BY CONTROL SCHEME WAS 50)
(GIVE CF (0 TO 100) THAT Y HAS SNOUT_IS_SHORT_AND_STRAIGHT IS
TRUE.) 20

SPECIFICITY_LEVEL_IS 10

DECISION_IS Y belongs to Genera Zaglossus.
(DO YOU WANT TO PROCEED FURTHER? WRITE Y/N.) y
(IT IS TRUE WITH CF = 76.52051)
(AND CF REQUIRED BY CONTROL SCHEME WAS 50)
T
Lisp> (exit)
```

REFERENCES

1. Michalski, R.S. and Winston, P.H., Variable Precision Logic, Artificial Intelligence 29 (1986) 121-146.

2. Winston, P.H., Artificial Intelligence (Addison-Wesley, Reading, MA, 2nd ed., 1984).

3. McCarthy, J., Circumscription- A form of non-monotonic reasoning, Artificial Intelligence 13 (1980) 27-39.

4. McDermott, D. and Doyle, J., Nonmonotonic logic I, Artificial Intelligence 13 (1980) 41-72.

5. McCarthy J. (1968), "Programs with common sense", in Semantic information processing, M. Minsky (ed.), Cambridge, Mass. MIT Press. (Article originally appeared in 1958).

6. Bonnet, A., Artificial Intelligence: Promise and Performance (PHI, UK., Ltd., 1985).

7. Becker, J.M., Inductive learning of decision rules with exceptions: methodology and experimentation, M.Sc. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, 1985.

8. Reinke, R., and Michalski, R.S., Incremental learning of concepts description, in: J.E. Hayes, D. Michie and J. Richards (Ed.), Machine Intelligence 11 (Oxford University Press, Oxford 1986).

9. Nguyen, H.T., Fuzzy Sets and applications: Selected papers by L.A. Zadeh, John Wiley & Sons, 1987 (April).

10. Winston, P.H., LISP: Adison-Wesly Publishing Company, 1984.

11. Steele, G. Jr., Common LISP: The Language, Digital Equipment Corporation, USA.

12. Bibel, W. and Jorrand, Ph., Fundamental of Artificial Intelligence, Springer Verlag, 1986.