LEARNING IN CENSORED PRODUCTION RULES SYSTEM : METHODOLOGY & IMPLEMENTATION

Dissertation Work Submitted to

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

R. SREENIVASAN

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI - 110 067
JANUARY 1991

To my parents and teachers

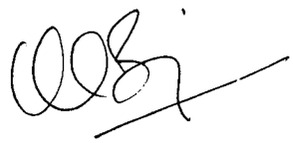
C E R T F I C A T E

This Dissertation titled

LEARNING IN CENSORED PRODUCTION RULES SYSTEM :
Methodology & Implementation

has been done by Mr. R. SREENIVASAN, 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.



Prof. K.K. Bharadwaj
School of Computer and Systems Sciences
Jawaharlal Nehru University
New Delhi.

Prof. N.P. Mukherjee
Dean, School of Computer and Systems Sciences
Jawaharlal Nehru University
New Delhi.

A C K N O W L E D G E M E N T S

I am grateful to my supervisor Prof. K.K. Bharadwaj, who got me interested in the field of learning and provided sagacious directions. His keen interest, inspiring guidance and constant encouragement through out the course of this work will be unforgettable for me. I fail to find words to express my gratitude to him for his genial advice, considered suggestions and veracious guidance that he gave benevolently.

I offer my sincere thanks to Mr Naveen Kr. Jain, Ph.D. student who helped me during this work.

I am indebted to the Dean, Prof. N. P. Mukherjee for all his help. I am very much thankful to all of my teachers, friends and collegues for their constant guidance, support and encouragement.

R. Sreenivasan

C O N T E N T S

Chapter One

Introduction

THE DISSERTATION - AN OVERVIEW

Introduction

Learning is an important Cognitive process. It denotes changes in the system that are adaptive, in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time.

The human ability to learn is truly remarkable though it is a long, slow process because of the inefficiencies of human information processing system in trying to extract patterns of other kinds of information from complex noisy situations and retaining those patterns in a manner that makes them available for later use.

Imparting learning capabilities in machines is one of the central goals of Artificial Intelligence. The penchant for discovering tricks that manage to escape the tediousness of human learning, however, provides a strong motivation for research in machine learning.

Fundamental Learning Strategies

Learning strategies are distinguished by the major type of inference the learning system performs on the information provided in order to derive the desired knowledge. At one extreme, system performs no inference, but directly accepts and uses the information given to it. At the other extreme, the system performs a complex, search-based inductive inference that on occassions leads to discovery of new knowledge.

The following learning strategies are important points along the above spectrum.

A. Direct implanting of knowledge

B. Learning by instruction

C. Learning by deduction

D. Learning by Analogy

E. Inductive learning

The order of these strategies reflects the increasing complexity of the inference performed on the information given to a learning system in order to derive the desired knowledge. Chapter 2 describes these strategies in detail.

Variable Precision Logic

Knowledge-Base is a major component of any learning system. It involves setting up of a correspondence between symbolic reasoning system and the outside world. The representation of knowledge should be comprehensible and should aid in implementing inference and reasoning process easily. In chapter 3 Knowledge representation is discussed.

Variable precision logic is concerned with the problems of reasoning with incomplete information and resource constraints. R.S. Michlaski and P.H. Winston [1], are the originators of this form of knowledge representation.

It attempts to develop mechanisms for representing and conducting reasoning that reflects different trade-offs between certainty (and/or specificity) of decisions and the computational resources needed to derive them.

Features

A. The representation enables rule repair mechanisms, there by modifying tentative knowledge because of changes in the real world.

B. It facilitates in giving precise or more certain results depending on the time one could spare. It is an extension of ordinary logic and enables logic-based systems to exhibit variable precision in which certainty varies while specificity stays constant.

Censored Production Rules

Winston and Michalski [1] employed censored production rules, which are production rules with exception to implement such a logic system. Intuition is that the censored production rules capture certain aspects of common sense knowledge which are absent from ordinary production rules. The system employing censored production rules handles only certainty part of the variable precision logic.

Ordinary production rule is of the form

IF < premise > THEN < action >.

Augmented production rule with exception is of the form

IF < premise > THEN < action > UNLESS < censor >.

where the censor is a disjunction of predicates; that when satisfied blocks the rule. The censor is an exception for the application of the rule.

This implementation enables to attain the said twin objectives.

A. When the real world changes, its symbolic representation can be changed by either generalising or specializing the production rule through censor. This leads to modifying the tentative database.

B. If a user cannot afford time, then the system can ignore the censor part while applying the rule, otherwise the censor part is looked into thereby resulting in more certain answers.

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 specificity and genarality information.

Hierarchical Censored Production rules

In order to incorporate variable specificity [19,20] we can augment the censored production rules with specificity part, like

IF <Premise>

THEN <Action>

UNLESS <Censor>

SPECIFICALLY <Specificity>

GENERALLY <generality>

Hence the decisions from such systems can exhibit variable specificity as well as variable certainty or reflect a trade-off between the two. The generality part of the rule aids in incorporating the learning aspect in a system which enables dynamic modification of the knowledge base. This forms the core of the dissertation. Chapter 3 discusses the HCPR system and learning process implemented in the system.

The Project

The aim of this project is to implement the learning process in censored production rules system from specific cases and precedents and to modify those rules incrementally to account for new facts. Then to extend the learning process to hierarchical censored production rules system. Chapter 4 discusses the implementation aspects of HCPRs and learning in such a system. A session with the system is also included.

Chapter two

Learning Strategies

LEARNING STRATEGIES

Introduction Learning is no doubt central to human intelligence. This ability permits us to adapt to the changing environment, to develop a great variety of skills, and to acquire expertise in almost an unlimited number of specific domains.

Implanting learning capabilities in machines is one of the central goals of Artificial Intelligence. With the advent of expert systems, implementing some forms of machine learning has become an urgent task, even if the forms of such implementation are very limited.

The major component of an expert system is its knowledge base, which is built through a cooperative effort between a knowledge engineer and an expert in the given domain of application. Such a system consists of production rules or a semantic network which represents knowledge and aid in implementing inference processes. Encoding expert knowledge into a system is a time-consuming, difficult process that is prone to error. For this reason knowledge acquisition is a 'bottleneck' in the development of expert systems. The process of knowledge acquisition can be simplified by applying interactive programming aids for developing and debugging rule bases. A long term solution, however, is seen in the development of machine learning.

Fundamental Learning Strategies

In this chapter some fundametal learning strategies are discussed - learning from instruction, learning by deduction, learning by analogy and learning by induction. Two basic types of learning by induction: learning from examples and learning by observation are discussed.

The knowledge acquisition process can be greatly simplified if an expert system can learn decision rules from examples of decisions made by human experts, or from its own errors. This type of learning is called learning from examples. Learning from examples is one of the fundamental learning strategies. The strategies are identified by viewing a learning system as an inference system. Namely, they are distinguished by the major type of inference the learning system performs on the information provided, in order to derive the desired knowledge. At one extreme, the system performs no inference, but directly accepts and uses the information given to it. At the other extreme, the system performs a complex, search-based inductive inference that on occasion leads to discovery of new knowledge.

The following learning strategies [14,15] are important points along the above spectrum :

Direct Implanting of Knowledge

This strategy requires little or no inference on the part of the learner. It includes rote learning, learning by imitation, learning by being constructed or by being programmed. This strategy is widely used method for providing knowledge to a computer system: we incorporate knowledge into its hardware, we program it, and we build databases for all kinds of applications. Although building databases is not typically considered as machine learning, it can be considered as a special case of such a process. Some databases go beyond this learning strategy, if they can perform some amount of inference, usually mathematical or statistcal.

Learning from Instruction

In this form of learning, also called learning by being told, a learner selects and transforms the knowledge from the input language to an internally-usable representation and integrates it with prior knowledge for effective retrieval and use. This is the most widely used strategy of human learning: it includes learning from teachers, books, publications, exhibits, displays and similar sources. A machine version of this strategy is a system capable of accepting instruction or advice and applying the learned knowledge effectively to different tasks.

Simple versions of this strategy constitute the basic method for providing knowledge to expert systems today.

Learning by Deduction

A learning system that uses this strategy conducts deductive (truth-preserving) inference on the knowledge it possesses and knowledge supplied to it. This is done in order to restructure given knowledge into more useful or more effective forms, or to determine important consequences of the knowledge. For example, given a set of numbers 1,2,6,24,120,720, a learning system might represent them in an equivalent, but shorter form as n!, n = 1..6. To do so, the system must, of course know the concept of a factorial.

A form of deductive learning, called analytical or explanation-based learning has recently become an active research area. In analytical learning, the system is already equipped with a description of the target concept, but the description is expressed at the level of abstraction too high to be directly usable. The system uses the domain knowledge to determine or explain why a given fact is an example of the concept. This process takes a form of formal proof, and produces a new concept description that is operational. This typically means that the concept is reexpressed in terms of properties used in the concept example.

The explanation-based learning is a useful technique, applicable to many problems. In order to be used, however, the system has to be equipped with a sufficient amount of relevant domain knowledge. This domain knowledge has to be inputed to the system somehow - either by handcrafting it into the system, or by anological or inductive learning.

Learning by Analogy

This strategy involves transforming or extending existing knowledge applicable in one domain to perform a similar task in another domain. For example, the learning-by-analogy strategy might be applied to learn water skiing when a person already knows snow skiing. Learning by analogy requires a greater amount of inference on the part of the learner than does learning from instruction. Relevant knowledge or skill must be retrieved from the memory and appropriately transformed to be applicable in a situation or to a new problem.

Inductive Learning

Inductive learning is a process of acquiring knowledge by drawing inductive inferences from teacher- or environmental-provided facts. This process involves operations of generalizing, transforming, correcting and refining knowledge representations in order to accomodate given facts and satisfy various additional criteria. An important property of inductive learning is that knowledge acquired through it cannot, in princple,

Inductive Learning Cont'd..

except for special cases, be completely validated. This is so because inductive inference produces hypotheses with a potentially infinite number of consequences, while only a finite number of confirming tests can be performed. Inductive inference is an underconstrained problem. Given any set of facts or input premises, one can potentially generate an infinite number of hypotheses explaining these facts. In order to perform inductive inference one thus needs some additional knowledge (background knowledge) to constrain the possibilities and guide the inference process toward one or a few most **plausible** hypotheses. In general, this background knowledge includes the goals of learning, previously learned concepts, criteria for deciding the preference among candidate hypotheses, the methods for interpreting the observations, and knowledge representation language with corresponding inference rules for manipulating representations in this language, as well as the knowledge of the domain of inquiry.

There are two aspects of inductive inference: the generation of plausible hypotheses, and their confirmation. Only the first is of significance to machine inductive learning. The second one is considered of lesser importance, because it is assumed that the generated hypotheses will be judged by human experts and

Inductive Learning Cont'd..

tested by known methods of deductive inference and statistical information.

The general paradigm of inductive inference is :

Given

a. **Premise statements**, F, represent initial knowledge

b. **A tentative inductive assertion**

c. **Background knowledge** (BK) that defines goal of inference, the preference criterion for ranking plausible hypotheses, assumptions and constraints imposed on the premise statements and the candidate inductive assertions, and any other relevant general or domain specific knowledge.

Find

an inductive assertion (hypothesis), H, that, together with background knowledge BK, tautolagically implies the premise statements.

An hypothesis together with background knowledge tautologically implies a set of facts, if the facts are logical consequence of the hypothesis and background knowledge, that is the implication **H & BK => F** holds under all interpretations. Since for a given BK an infinite number of assertions H can satisfy such an implication, a **preference criterion** is a used to reduce the choice to one hypothesis or a few most preferable ones. Such a criterion may require, for instance, that

the hypothesis be the shortest or the most economical description of all given facts, among all candidate descriptions.

The two important Inductive learning strategies discussed are Learning from examples and learning by Observation and Discovery.

Learning from Examples

Given a set of examples and counter-examples of a concept, the learner induces a general concept description. The amount of inference performed by the learner is greater than in learning by deduction or analogy, because the learner does not have prior knowledge of the concept to be learned, or knowledge of similar concept. Thus, it cannot create the desired knowledge by deduction, or by analogy to what it already knows. The desired knowledge must be created anew by drawing inductive inference from available examples or facts. Learning from examples, also called **concept acquisition**, can be a one-step process or multi-step process. In the former case, all examples are presented at once. In the latter, examples are introduced one-by-one or in small groups; the learner forms one or more tentative hypotheses consistent with the data at a given step, and subsequently refines the hypotheses after considering new examples. This strategy is commonly used in human learning.

Learning by Observation

This 'learning without teacher' strategy includes a variety of processes, such as creating classifications of given observations, discovering relationships and laws governing a given system, or forming a theory to explain a given phenomenon, or the learner is not provided with a set of instances exemplifying a concept, nor is given access to an oracle who can classify internally-generated instances as positive or negative. Also, rather than concentrating attention on a single concept at a time, the learner may have to deal with observations that represent several concepts. This adds a new difficulty, namely solving the focus-of-attention problem, which involved deciding how to manage the available time and resources in acquiring several concepts at once.

Learning from observation can be subclassified according to the degree of interaction between the learner and the external environment. Two basic cases can be distinguished:

a. **passive observation** where the learner builds a description of a given set of observations.

b. **active experimentation,** where the learner makes changes in the given environment and observes the results of those changes. The changes may be random or dynamically controled by some heuristic criteria.

These learning strategies were presented above in order of increasing amounts of effort required from the learner and decreasing amounts of effort required from the teacher. This order thus reflects the increasing difficulty of constructing a learning system capable of given learning strategy.

Chapter three

Knowledge Representation

KNOWLEDGE REPRESENTATION

Introduction

In order to solve the complex problems encountered in artificial intelligence, one needs both a large amount of knowledge and some mechanisms for manipulating that knowledge to create solutions to new problems.

A variety of ways of representing knowledge have been exploited in A.I. programs. Some of them, Two-valued logic - syllogistic logic, symbolic logic, propositional logic, predicate logic; multivalue logic, nonmonotonic logic, probabilistic logic, fuzzy logic and variable precision logic are discussed here.

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, dead or alive, male or female and so forth are classes that have sharp boundaries.

Syllogistic Logic

The first known logician was Aristotle (384-322 B.C.), the great philosopher 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 from a philosophical argument. This type of logic is still used because it is the basis for virtually all legal argumentation. example

X --> Y; All Y --> Z hence X --> Z

In many ways, syllogistic logic is simply a formalisation of common sense. However, because syllogistic logic is based in natural language, 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.

Symbolic Logic

Symbolic logic began with G.W. Leibniz (1646-1717), but was forgotten when he died. The entire field was revived 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)

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

Propositional Logic

It deals with the determination of 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 logic.

Predicate Logic

This is an extention of propositional logic. The basis of predicate logic is predicate, which is essentially a function that returns either a value of true or false depending upon its argument.

example 1: The predicate Dog defined by Dog(X) takes the value True if X=Rover False if X=Pussy

example 2: is_hard (rock) ==> true

is_hard (cotton) ==> false

In propositional logic these two predicates and arguments become rock is hard cotton is not hard. In predicate logic, it is possible to create a function that determines the hardness of any object. The predicate logic uses variables to generalise predicates Man(X) ==> Not woman(X).

predicate logic uses two quantifiers, existential (there-exists) and universal (for-all). example -

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)))

Multivalue Logic

A polish mathematician J Lukasiewiecz, first developed the concept of multivalued logic during 1920s. In multivalued logical systems, there are more than two truth values. There may be 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.

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 complexion among swedes is blond. These examples illustrate one common kind of default reasoning, which may be called most probable choice.

Probabilistic Logic

Probobilistic 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:

a. 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.

b. 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.

c. The world appears to be random because we have not described it at the right level, e.g. characters when analysed as lines and archs appears to be less randomly varying than when viewed as a collections of dots.

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 tools", "a tall man", etc.

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 member ship 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.

Variable Precision Logic

In 1986, Michalski and Winston in their paper [1], introduced the censored production rules to exhibit variable precision logic. Variable precision logic is concerned with reasoning with incomplete information and resource constraints. It offers mechanism to handle trade-offs between the precision of inferences and the computational efficiency of deriving them. You cannot tell an ordinary logic-based reasoning system much about how you want it to do its job.

You cannot give the following instructions, for example

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

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

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

d. 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.

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 certainty are inversely related. Variable precision system is a system that exhibits either variable specificity or variable certainity or some trade-off between the two.

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 predictes 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 assertions of the form:

"Premise ===> decision"

But this is not always the case. Ther 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:

a. The **rule base**, set of production rules.
b. The **fact base**, definitions & data structures containing known facts.
c. 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 B lays eggs and B flies then B is a bird

Fact base: F1 Buzo flies

F2 Buzo lays eggs

Fact base after the interpreter has scanned both facts and rules F1 Buzo flies

F2 Buzo lays eggs

F3 Buzo is a bird (new fact derived)

A production rule lacks certain aspects of common sense knowledge. Precision of inferences remains constant in production rules system, since neither certainty 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, 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. The last two 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.

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.

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 action Unless censor." The censor is a logical condition (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 exceptions, using censored production rule is a good choice.

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 censored production rule, while exhibiting more intelligence. The resulting rule will be called a hierarchical censored production rule or simply HCPR.

Chapter four

Hierarchical Censored Production Rules and Learning

Hierarchical Censored Production Rules

In a system during problem solving, at any instance, if information about the applicable rules is provided then the process will be fast, because the system need not find them using exhaustive search of the whole rule base. e.g. A query asked - 'can Buzo fly?'; the process after finding that 'Buzo can fly', should look for the next line of action to be taken by the system to get more specific answer for 'can fly how high?' or 'can fly how fast?' rather than search the rule base and coming across irrelevant rules. 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 and learning.

Such a behaviour exhibited by system should be regarded as "intelligent behaviour", 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. Augmenting censored production rule with information of rules, which should apply next to get more specific decision results in Hierarchical Censored Production Rule (HCPR).

HCPRs

A HCPR is of the form

If <premise>

Then <action>

Unless <censor>

Specifically <specificity>

Generally <generality>

It is created by augmenting the censor production rules with the specificity and generality information. 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.

The generality information to a rule is

a. to give the parent of the current rule in the rule tree.
b. to help in the process of learning in HCPR system by providing a trace to the system in finding the parent for the new 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.

RULE-TREE A rule-tree is a collective and systematical representation of all related HCPRs about a given 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 domains

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 rule-tree cannot represent more general concept than the 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 more general rules.

Rule-tree Cont'd..

Rules dependent on same immediate general rule are called sibling rules and the general rule is called the parent rule. Each rule has only one parent rule.

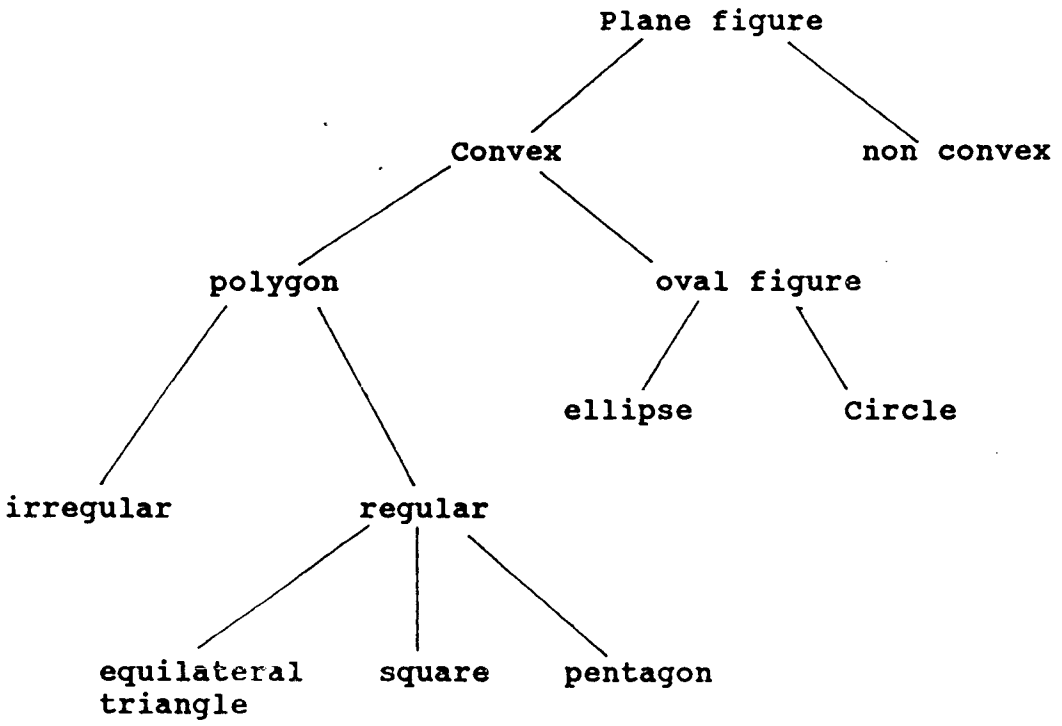
The advantages of rule-tree

a. The rule-tree offers mechanisms to handle various trade offs
   1. It offers mechanism to handle trade-offs between the precision of inferences and computational effeciency of deriving them.
   2. It offers mechanism to handle trade-offs between the certainity of conclusion and its specificity.
b. The representation using rule-tree facilitates efficient use of memory.
c. The rule-tree provides an efficient mechanism to discard most of the task irrelevant information provided to an intelligent system. It considers task relevant information only.
d. 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 and hence, it is not required to consider the whole knowledge-base.

Rule-tree Cont'd..

e. The generality information coupled with specificity information aids the process of learning in the system. The system traverses through the rule-tree to locate the parent of the entered new rule. There by strengthening the rule-tree and dynamically updating the rule-tree. The newly entered rule may be a sibling of the tracked rule, parent of the tracked rule or it may be sibling of one rule and parent of another rule depending the rule's generality and specificity characteristics.

So it is superior to other systems because it requests information that has the greatest importance, 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 uncertainity from the system.



The rule-tree in this figure is for the concept of plane-figures. The root of the tree represents a general concept of plane figure, and its subtree with root convex represents 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 plane figure, convex and polygon.

LEARNING

Learning is a continuous process and results in the modification of the system to improve systems' behaviour. The performance of the system increases with time due to the dynamic modification of the knowledge-base. Because of this fluidity of knowledge, the representation of knowledge should aid modification of knowledge-base.

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 exception, specificity informations may be incomplete or even absent in a HCPR depending on, how much a system has learnt. Some of the following learning schemes are implemented in this project.

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 most 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.

Learning by refining beliefs

Strength of implications and likelihoods 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 :

a. Number of times a HCPR has been employed successfully
b. Number of times a particular exception has blocked the rule.

Learning by Concept Formation

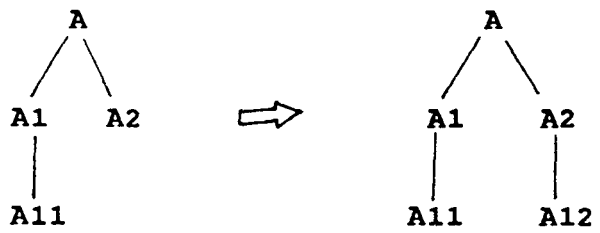
A concept corresponds roughly to an idea; a concept is introduced into the system by way of definitions which gives its relations to other concepts in the system.

One important use of concepts is to provide the informational bases for classifying newly encountered objects or events. A concept may be defined as ordered information about the properties of one or more things that enables any particular thing or class of things to be differentiated from and also related to other things or classes of things.

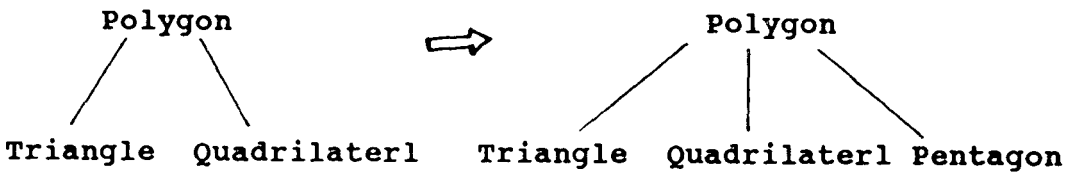
A system should have learning algorithms which may be used to neutralise the errors in a concept due to over generalisation, under generalisation, or misconception. These issues of concept formation are described by naming different operations on rule-tree appropriately.

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 horizontal growth, as shown in the figure.



General Concept

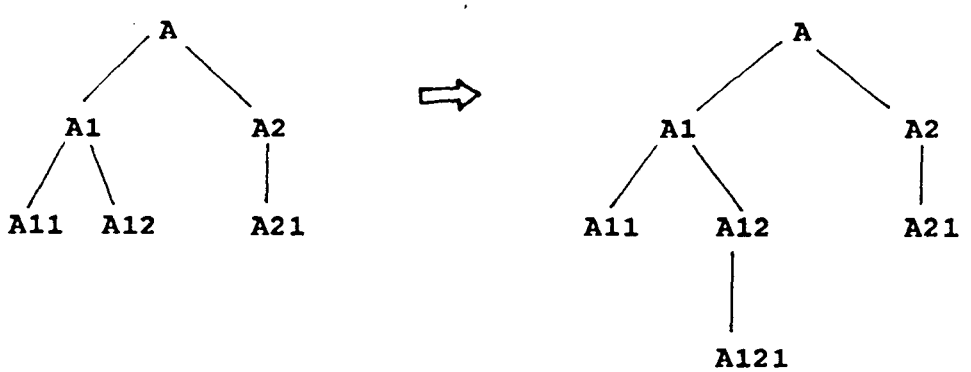


Polygons

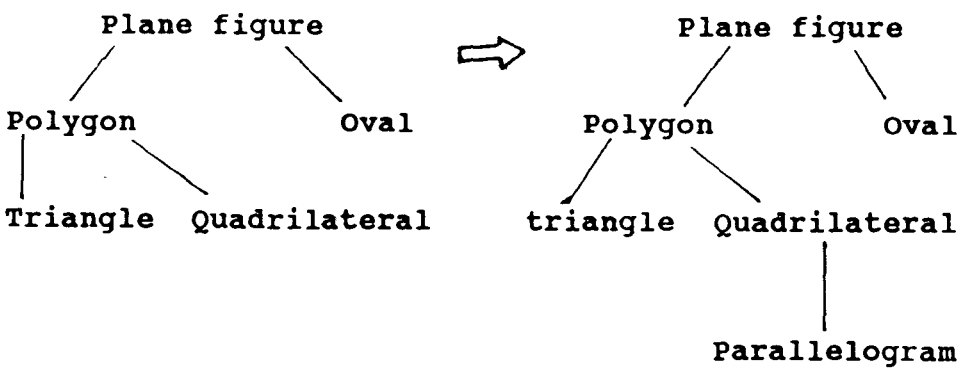
Horizontal growth of a hcpr-tree for general concept and polygons

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 vertical growth, as shown in the figure.



General Concept

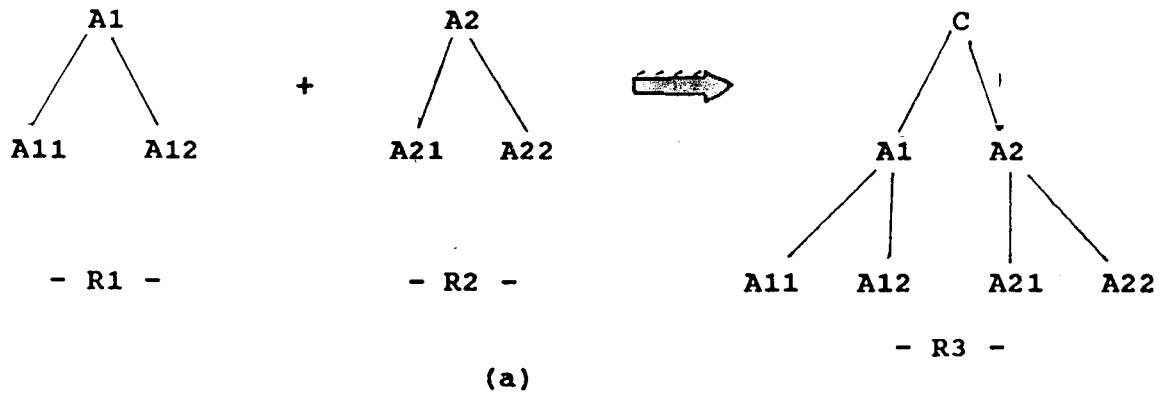


Plane figures

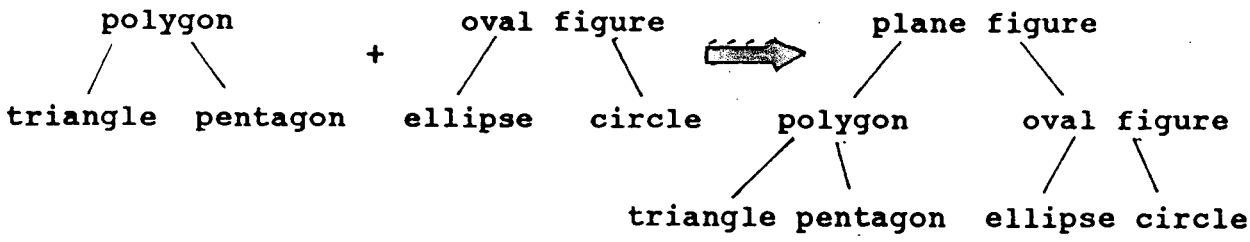
Vertical growth of a hcpr-tree for general concept and plane figures

Learning by Fusion

a. Consider two rule-trees namely R1 and R2 which represent two unrelated concepts "A1" and "A2" initially. But, after some time it is observed that these two concepts are related such that, they may be combined to form a single concept, say C. A new rule-tree is then formed to represent the general concept C as root and rule-trees for decisions R1 and R2 as its subtrees, as shown in the figure.



(a)



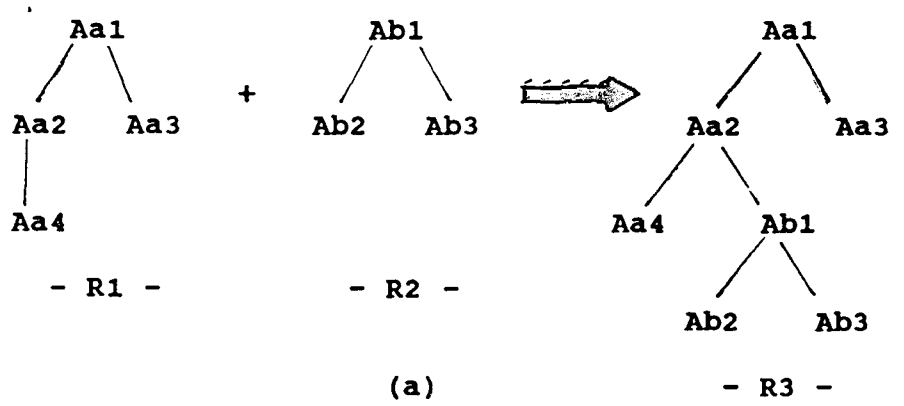
(b)

Fusion of two hcpr-trees for

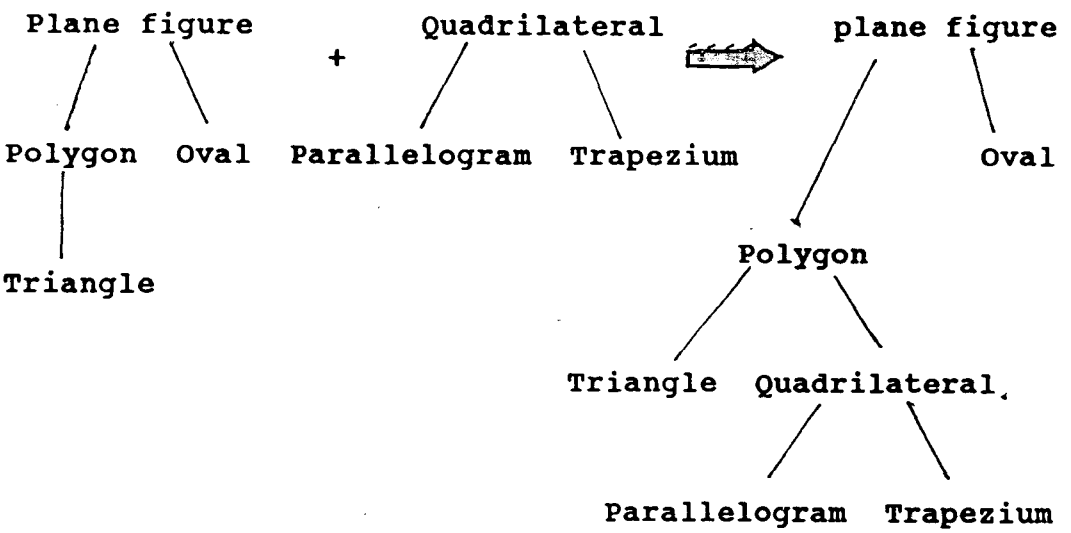
(a) general concepts (b) polygon and oval figure

Learning by fusion

b. A rule-tree R2 with an independent concept "Ab1" initially, may be a subconcept of another rule-tree for concept R1. So after suitable modifications a relationship is established between the two rule-trees, as shown in the Figure.



(a)



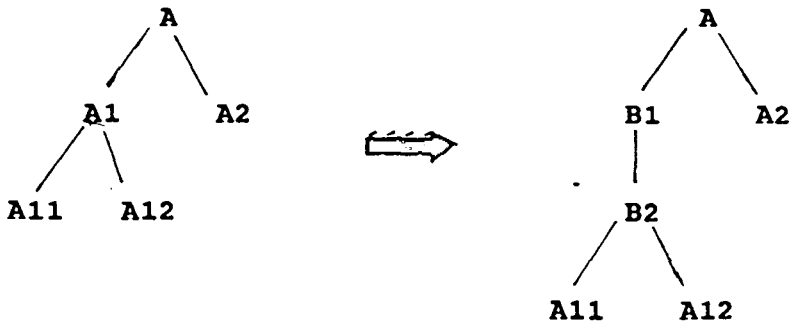
(b)

Fusion of two hcpr-trees for

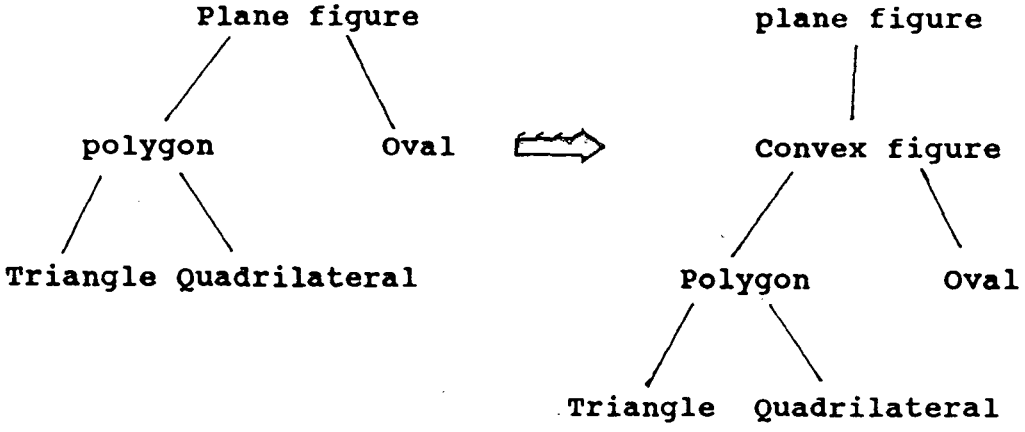
(a) general concepts (b) plane figure and quadrilateral

Learning by Vertical 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, as shown in the Figure. This process is called vertical fission.



(a)

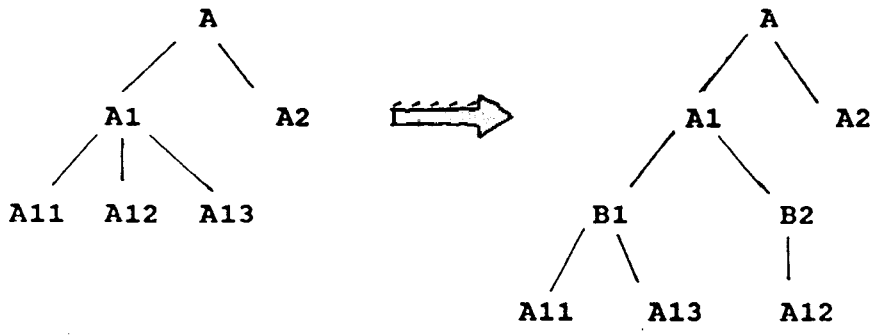


(b)

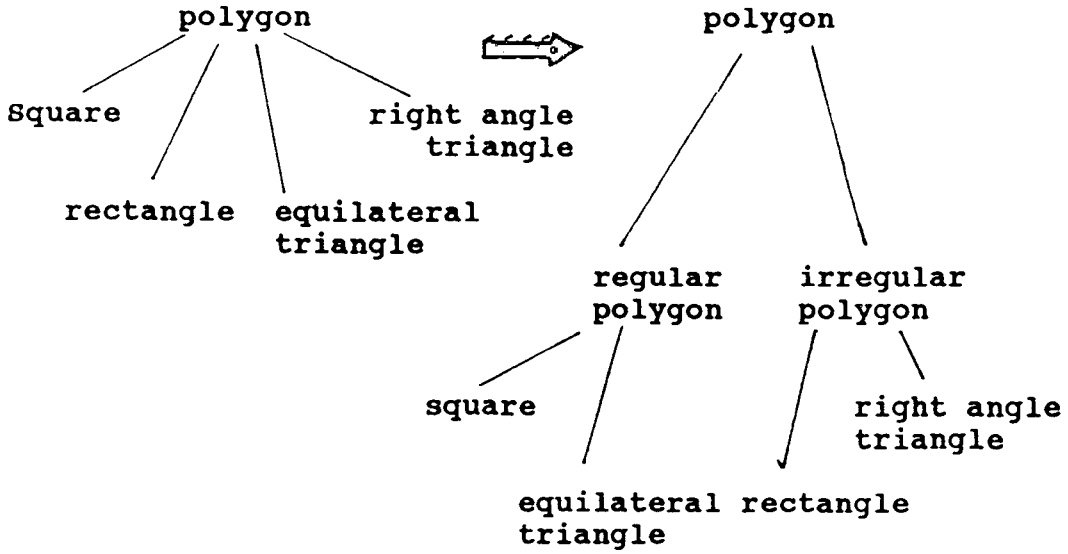
Vertical fission of a hcpr for
(a) general concept (b) plane figure

Learning by Horizontal Fission

As the number of sibling rules crosses some critical value, they may split under two or more intermediate parent rules, as shown in Figure. Where, B1 and B2 are two intermediate parent rules. This prosses will be called horizontal fission.



(a)



(b)

Horizontal fission of a hcpr-tree for

(a) general concept (b) polygon

Chapter five

Implementation

Computer System Used

The System used is IBM compatible BUSYBEE 386. It is a 32-bit Personal Computer. It is compact, powerful, general purpose micro-computer which has been set to suit most of the personal and business computing needs.

It is based on INTEL 80386 32-bit microprocessor. A large variety of popular application software packages like Symphony, Lotus 1-2-3, Softword, dBASE, and compilers for Fortran, Basic, Cobol, Pascal, C, LISP and Prolog languages etc. run on this system in DOS environment.

The system can also work in multiuser Xenix environment and can support upto eight users. Various popular packages can run in this environment.

The system has one floppy disk drive, one cartridge drive and one hard disk drive. The hard disk provides more storage capacity, greater efficiency and faster data access than the floppy disk drive. The hard disk has 300MB storage capacity and floppy disk has 1.2 MB capacity.

It has RAM configuration of 8192 K bytes. It also has a graphic adapter.

System Configuration

The basic components of the system are as follows:

The System Unit

This comprises of disk drive and microprocessor unit. The microprocessor used is INTEL 80386 32-bit.

Storage Devices

This has a floppy drive of 1.2MB capacity and one hard disk with 300MB of disk storage capacity. In addition to this there is a cartridge drive meant for back up of softwares.

Video Display Unit

The system can display its output on a monochrome video monitor. This is connected to the system through a power cable.

Keyboard

The system has QWERTY keyboard. The keys are low profile, full travel, sculptured type. The keyboard can be inclined for typing comfort.

Graphics Adapter

The monochrome/color graphics adapter is an add-on card to which you connect your monochrome or color monitor. The adapter supports two basic operation modules - text and graphics. In the text mode, one can have eight pages of 40 x 25 text or four pages of 80 x 25 text with attributes such as Reverse Video, Blinking and Highlighting. With the color monitor we can have upto 16 different foregrounds and 8 background colors.

The text mode can display the 256-character set that includes 16 special characters for games, 15 character for editing, 96 ASCII set, 48 foreign language characters, 16 Greek characters and 15 scientific notations.

In the graphics mode, one can have a resolution of 640 x 200 pixels with monochrome monitor or 320 x 200 pixels with color monitor.

Printer

The printer used is EPSON FX - 105. This is fast, reliable and easy to use. It has a number of print modes. For fast draft printing, we can choose to print at 160 characters per second. For higher quality printing, using NLQ mode, the speed is reduced. This printer can take either cut sheets or continuous stationery. The appearance of the printer page is controlled by the codes sent by the computer to the printer. We can use codes ourselves to produce an enormous range of special printing effects.

OPERATING SYSTEM

An Operating System is a set of commands which help the users to share a computer installation efficiently. An operating system is silent partner when we are using the computer. It provides the interface between the hardware and both the user and other system software.

An Operating System (OS) is a piece of system software most closely associated with the hardware. The OS is unique to the microprocessor. For example, MS-DOS runs on the microprocessor 80x86 family and will not run on another microprocessor like z8000 unless major parts of the OS are rewritten.

The operating system used is MS-DOS. Microsoft MS-DOS is a disk operating system for INTEL chips based computers. Through MS-DOS we communicate with the CPU, disk drives, printer, and manage these resoursces to our advantage. It enables us to create and keep track of files, link and run programs, and access peripheral devices such as printer and disk drives that are attached to our computer. A whole range of softwares and compilers are available under MS-DOS.

Software for Artificial Intelligence

Artificial Intelligence, or AI, is the branch of computer science concerned with making computers "intelligent." This brings the fundamental difference between traditional programming and ÁI programming [18]. In traditional programming, the program represents knowledge. The programmer has thought out the procedures and steps to solve a problem and the program therefore knows how to solve it. In contrast, AI based programming does not represent knowledge as part of the program procedures. Instead, it keeps knowledge and control separate and makes the program act as the interpreter. This allows the user to modify or change the knowledge domain and solve different problems without having to change the program itself.

Features

Language features that are particularly important in building AI systems are -

* Good facilities for manipulating lists
* Pattern matching facilities, both to identify data and determine control.
* Facilities for performing some kind of automatic deduction and for storing a database of assertions that provide the basis for deduction.
* Facilities for building knowledge structures, such as Frames, so that related pieces of information can be grouped together and accessed as a unit.

* Mechanisms by which the programmer can provide additional knowledge that can be used to focus the attention of the system where it is likely to be the most profitable.
* Control structures that facilitate goal-directed behavior in addition to the more conventional data-directed processing.
* The ability to intermix procedures and declarative data structures in whatever way best suits a particular task.

LISP

LISP is by far the most important member of the AI language family, in terms both of the number of lines of code written in it and its influence on the development of other languages. LISP was first presented by McCarthy as notation for defining mathematical functions. But quickly became the favoured language for AI systems.

There are several reasons why LISP is a good language in which to build AI systems [13] :

* Its principle data structure is the list, which is very useful in representing much of the knowledge used in AI programs.
* A collection of facts about an individual object can easily be represented in the property list that is associated with the atom representing the concept.
  The most natural control structure is recursion, which is appropriate for many problem-solving tasks.

LISP

* Bindings of most things occur at the last possible moment. For example, list need not be of fixed size, and the set of known properties can change dramtically as a program executes.

* The fact that both data and procedures are represented as lists makes it possible to integrate declarative and procedural knowledge into a single structure such as a property list. It also makes it possible for a program to construct a procedure and then execute it.

* Most LISP systems run interactively. This facilitates the development of all kinds of programs. It also makes it possible to write truly interactive programs. This is often very important in AI application areas where the problems are too hard to be solved by a program without human intervention.

There exist many dialects of LISP, varying on everything from the names of standard functions and order of their arguments to substantive issues involving the kinds of features provided. The LISP used in this project is Common Lisp.

A representation is a convention that constitutes or determines the relation between a representation and that which it represents. Problem and Knowledge representation is the key to a successful design of AI software or any problem-solving techniques. The first task in solving any problem is to map the real problem into its representation, which can provide us with an elegant solution. This process involves selecting an appropriate data structure or medium for the representation and an appropriate way to represent the problem in the selected medium.

The choice of knowledge representation depends on the functions one wishes to perform. Knowledge representation is one of the most difficult parts of the problem-solving process. It should be well thought out and planned. No methodology exists for designing and developing a representation of a problem at hand. However, over the years, many approaches for representing problems and knowledge have been devised. But still representation has been left to the imagination, creativity, and experience of the person who attempts to tackle the problem.

The first step in designing and implementing a knowledge representation mechanism is to select a medium for storing relations among objects.

Production Rules

Production systems provide an excellent tool for structuring AI programs. Production systems create an environment or shell for applying production rules,or alternatively, collections of production rules designed to serve some specific purpose. Production rules also are called 'antecedent-consequent rules', 'premise-action pairs'. As discussed in the last chapter the learning and reasoning process is implemented using Hierarchical Censored Production Rules which are 'premise-action-censor-specificiy-generality rules'. This representation is implemented using Frames.

Frames

The idea of frames was originally proposed by Marvin Minsky in 1975 in his paper 'A Framework for Representing Knowledge,' in Psychology of Computer Vision, McGraw Hill, 1975. Minsky presents frames as building blocks for understanding visual perception, natural language dialogues, and other complex phenomena. A frame is the data structure appropriate for representing a stereotypical situation. Frames organize knowledge into prototypical objects and stereotypical events that are appropriate for specific situations. Psychological evidence suggests that people use large frames for encoding knowledge from their previous experiences, or knowledge about commonly encountered things, in order to analyze and explain a new situation in their everyday cognitive activity.

Syntax of Frames

```
( [frame name]
        ( [slot1 name]
                ( [Link1 name] value )
                ( [Link2 name] value )
                        .
                        .
        )
        ( [slot2 name]
                ( [Link1 name] value )
                ( [Link2 name] value )
                        .
                        .
        )
        .
        .
)
```

A frame consists of its name, followed by one or more slots. Each slot can hold one or more links. Each link has an associated value.

In the system, the HCPRules are represented as frames -

```
( rule-name ( if        (antecedent1)
                        (antecedent2)
                            .
                            .
            )
            ( then      (consequent1)
                        (consequent2)
                            .
                            .
            )
            ( unless    (censor1)
                        (censor2)
                            .
                            .
            )
            ( specific  (specificity1)
                        (specificity2)
                            .
                            .
            )
            ( general   (generality)
            )
)
```

All rules in the system are of this form. Each rule has all these five slots if, then, unless, specific, general,but the links may be empty.

Example :

```
(fly? (if         ((> subject) is a (> type))
                  ((< type ) is a bird)
      )
      (then       ((< subject) can fly)
      )
      (unless     ((< subject) is a special bird)
                  ((< subject) is not well)
      )
      (specific   (fly_high?)
                  (fly_fast?)
      )
      (general)
)
```

here the link for slot 'general' is empty, hence this rule is the root of the HCPR rule-tree. At most there can be only one link in general slot, because a rule can be a sibling for only one rule. If the rule is a terminal node in the rule-tree the specificity slot will have no links. Same is the case for unless slot, if there are no censors. The links for slots 'if' and 'then' cannot be empty, otherwise the rule becomes void.

The lisp interpreter is loaded from DOS prompt by typing the command 'lisp'. Then LISP prompt Lisp > appears. Now the system is loaded typing (load cprules) as -

Lisp > **(load cprules)**

(loading cprules...) appears on the screen. After loaded type (cprules) for initating the system as -

Lisp > **(cprules)**

The main menu appears on the screen immediately -

Main Menu

1. Interrogate the system

2. Add a rule to the knowledge base

3. Modify a rule in the knowledge base

4. Remove a rule from the knowledge base

5. Add a fact to the data base

6. Modify a fact in the data base

7. Remove a fact from the data base

8. Periodic update of the system

9. Quit the menu

Enter the choice : ÉÉ

The user may enter the choice as he wishes. We now go through all the choices.

Choice 1. **Interrogate the system**

On choosing the option, the system immediately goes into interactive mode as

(Enter the name of the subject)

(Buzo)

(What would you like to know about Buzo ?)

(Can Buzo fly?)

(Rule Fly? says Buzo can fly)

(Rule Fly_high? says Buzo can fly high)

(Rule Fly_fast? says Buzo can fly very fast)

(Ok it is done)

Press any key to continue ...

on pressing any key the main menu is again displayed. The result of this interaction is the three obtained facts about Buzo, even though the user wanted to know only whether Buzo can fly. The last two conclusions obtained by the sytem are due to the specificity part of the rule which checks whether 'Buzo can fly' from the available facts in the data base. This recursive process of diving down the rule tree can be curbed by specifying the certainty factor required for the result. This will be checked with the certainty factors of the consequents in the rules of the rule base which may be entered when the rule was added to the knowledge base. This helps the system to exhibit variable certainty and variable specificity.

Another query -

(Enter the name of the subject)

(sweekums)

(what would you like to know about sweekums)

(can sweekums fly?)

(Rule fly? says sweekums cannot fly)

Press any key to continue ...

Here the censor in the rule has obstructed the rule because data base has a fact which says 'sweekums is a special bird' which obstructs the rule.

Similarly if no information on a subject exists in the data base system gives the message

(Sorry, No information exists about Nancy)

(Enter if you know anything by choosing the choice)

Press any key to continue ...

By choosing the appropriate choice in the main menu, information about Nancy can be added to the data base.

How the system works

* System takes subject and rule during interaction
* Searches for the rule in the knowledge base
* Substitutes the available values for variables in the antecedents.
* Creates an associate list by binding the pattern variables to the matched values.
* Checks whether censor blocks the rule by substituting the values of variables in censors and matching facts
* If not blocked goes down the tree repeating the steps.

Choice 2 **Add a rule to the knowledge base**

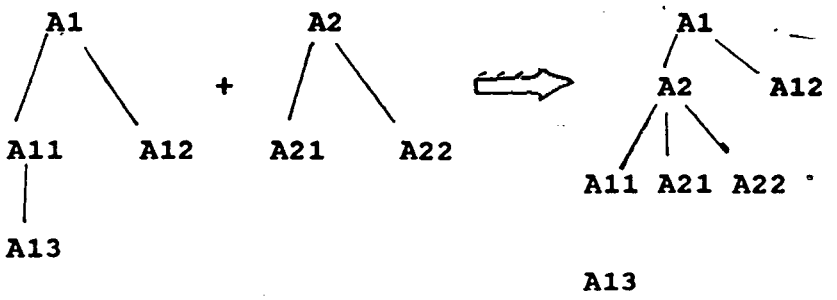
On selecting the choice the system goes into interactive mode -

```
(Enter the name of the rule to be added)

(fly_high?)

(Enter the Antecedents)
IF :
(((> subject) is a (> type))((< type) is a bird)
((< subject) has big wings))

(Enter the Consequents)
Then :
(((< subject) can fly high))

(Enter the Censors)
Unless :
()

(Enter the Specificity)
Specific :
()

(OK it is done)

(The new rule you added becomes)
(fly_high? (if      ((> subject) has big wings))
           (then    ((< subject) can fly high))
           (unless)
           (specific (fly_very_high?))
           (general (fly?)) )
```

The rule displayed seems to be different from the one we added, this is because of the learning process the system undergoes. The system has traced the appropriate location where the rule should lie and makes the requisite changes in the Knowledge Base. This is explained in the next page.

Choice 2 Cont'd..

A rule-tree A1 existed in the knowledge base and the rule A2 is added the system undergoes a learning process and new rule-tree results as follows



This is discussed in the last chapter as fusion.

The rule-tree had a parent and child as following rules

```
(fly? (if       ((> subject) is a (> type))
              ((< type) is a bird))
      (then     ((< subject) can fly))
      (unless   ((< subject) is a special bird)
              ((< subject) is not well) )
      (specific (fly_very_high?))
      (general))

(fly_very_high? (if   ((> subject) has big wings)
                      ((< subject) has powerful wings) )
                (then ((< subject) can fly very high))
                (unless ((< subject) is a heavy bird))
                (specific (fly_altitude?))
                (general (fly?)) )
```

After the new rule **fly_high?** is added the three rules look as follows due to the fusion the rule-tree undergoes

```
(fly? (if       ((> subject) is a (> type))
              ((< type) is a bird))
      (then     ((< subject) can fly))
      (unless   ((< subject) is a special bird)
              ((< subject) is not well) )
      (specific (fly_high?))
      (general))
```

```
(fly_high? (if      ((> subject) has big wings))
           (then    ((< subject) can fly high))
           (unless)
           (specific (fly_very_high?))
           (general (fly?)) )

(fly_very_high? (if   ((> subject) has powerful wings) )
                (then ((< subject) can fly very high))
                (unless ((< subject) is a heavy bird))
                (specific (fly_altitude?))
                (general (fly_high?)) )
```

observe how the slots if, specific and general have been changed during the fusion process.

How the system works

* System accepts the rule and looks out for a rule, whose antecedents is the subset of that of the new rule. If available, then the rule is a general form of the new rule.

* The system searches for the root of the rule-tree recursively with the help of the general slot of each rule. When the recursion unwinds, the system eleminates antecedents of the general rules from the new rule and descends down the tree till it is at starting node.

* The system goes down the rule-tree from the starting node with the help of specificity information and by comparing the antecedents of the rule at the node and that of the new rule. If antecedent of the node is subset then still dive down, else check whether other wise. If true then the new rule is the parent of the rule at the node. With help of the procedure fusion the system rearranges the rule-tree. If both possibilities are not valid then try for all other specificities of the rule till a proper place is found and do either fusion or fission as the need may be, by calling the appropriate procedures.

Choice 3 **Modify a rule in the Knowledge Base**

On choosing the option, the system enters interactive mode -

```
(Enter the name of the rule you want to modify)

(fly?)

(The rule fly? is)

(fly? (if       ((> subject) is a (> type))
                ((< type) is a bird))
      (then     ((< subject) can fly))
      (unless   ((< subject) is a special bird)
                ((< subject) is not well) )
      (specific (fly_high?))
      (general))


(What do you want to modify in the rule fly?)

(unless)

(Enter the new value of unless for rule fly?)

(((< subject) is a special bird))

(The modified rule fly? is)

(fly? (if       ((> subject) is a (> type))
                ((< type) is a bird))
      (then     ((< subject) can fly))
      (unless   ((< subject) is a special bird))
      (specific (fly_high?))
      (general))

(Ok it is done)

Press any key to continue ..
```

How the system works

* Takes the rule name, checks for existence, displays.

* Takes the slot, accesses the slot in the rule.

* Takes the new value and modifies the slot of the rule.

Choice 4 **Remove a rule from the knowledge base**

On choosing this rule the system slips into interactive mode -

(Enter the name of the rule you want to remove)

(fly?)

(ok it is done)

Press any key to continue ..

How the sytem works

* The system accepts the rule name, verifies existence.
* If present removes from the knowledge base.

Choices 5, 6 and 7 are for adding, modifying and removing facts from the facts base. Choice 8 is for periodic updation of data base and knowledge base. The data base is updated through forward chaining of the rules on the facts. The knowledge base is updated by going through the Knowledge base recursively to ascertain the position of each rule in the rule-tree and effecting the relevant modification.

CONCLUSION

A system should effect modification in the data base and knowledge base when it comes across a change in the environment which is represented in the system. This modification can be effected only when knowledge is represented in an effective manner and only when the system is able to percieve the change in the environment and learn to adapt to the change.

Hence learning strategies must be incorporated in the system. Various popular learning strategies are discussed. In order to aid the process of learning, knowledge is represented in an effective manner through hierarchical censored production rules. It has the ability to incorporate into the knowledge structure additional information that can be used in the process of learning and inference in promising manner.

The learning strategies that HCPR system can support are discussed. The learning by concept formation is implemented. Of these, fission and fusion hold good prospect.

The implementation of these two concepts, though encouraging, could not be realised to the full extent because of some limitations of the developed system. The domain is small with only specific cases being considered. However, based on the ideas developed here, further improvement of the system would be possible, so as to realise full potential of the HCPR representation of the knowledge, in the process of learning.

Appendix

```
;procedure  for  initiating the system, loads the knowledge  base  and
;from  the disk and displays the mainmenu, waits for the responce  and
calls the appropriate procedure to carry out the task
;calls procedures enquire, addrule, changerule, removrule, addfact,
;                 removefact, changefact, update
(DEFUN cprules ()
  (init)
  (setq facts (open_file (fn) ':io))
  (read_facts facts)
  (setq rules (open_file (fn1) ':io))
  (read_rules rules)
  (DO ((m (main_menu) (main_menu))
(s (select 9) (select 9)))
       ((= s 9) (quit))       ;repeat until user quits
   (COND ((= s 1) (enquire  "KNOW ABOUT")) ; retrieve
 ((= s 2) (addrule))   ; add a new rule
 ((= s 3) (changerule))    ; change the rule
 ((= s 4) (removrule))         ; remove the rule
 ((= s 5) (addfact))   ; add a new fact
 ((= s 6) (changefact))    ; change fact
 ((= s 7) (removefact))    ; remove a fact
                ((= s 8) (update))                ; periodic update
   )
   (press_a_key) ;wait
   ))

;procedure for displaying the main-menu of the system
;called from the procedure cprules
(DEFUN main_menu ()
    (clear-screen)
(set-cursor 1 29)(princ "-<< MAIN MENU >>-")
(set-cursor 5 20)
(princ "1. Interrogate the system")
( set-cursor 7 20)
(princ "2. Add a rule to the knowledge base")
( set-cursor 9 20)
(princ "3. Modify a rule in the knowledge base")
( set-cursor 11 20)
(princ "4. Remove a rule from the knowledge base")
( set-cursor 13 20)
(princ "5. add a fact to the data base")
( set-cursor 15 20)
(princ "6. Modify a fact in the data base")
( set-cursor 17 20)
(princ "7. Remove a fact from the data base")
( set-cursor 19 20)
(princ "8. Periodic update of the system")
       ( set-cursor 21 20)
       (princ "9. Quit the menu")
'main-menu
)
```

```
;A section of the procedure addrule which uses the procedures position
;to  locate the appropriate location for the rule, modifies  rule-tree
;using fission and fusion procedures
;called from the procedure cprules

  (SETQ alter (position ifspos adrul))
  (COND ((= (LENGTH alter) 1) (SETQ rules (CONS newrule rules)))
        ((EQUAL (LAST alter) '()) (RPLACD (ASSOC if newrule)
                                          (CAR alter))
                  (RPLACD (ASSOC general newrule)
                  (LIST (CADR alter)))
         (SETQ rules (CONS newrule rules))
                  (fusion (CADR alter) (LIST adrul)))
        (T (RPLACD (ASSOC if newrule) (CAR alter))
         (RPLACD (ASSOC general newrule) (LIST (CADR alter)))
                  (RPLACD (LAST (ASSOC specific newrule)) (CDDR alter))
                  (SETQ rules (CONS newrule rules))
         (fission (CADR alter) (LIST adrul) (CADDR alter)) );T
   )
   (PRINT '(The new knowledge base is))
   (PRINT (ASSOC adrul rules))
   (ok)

;This procedure finds out the node to which the incoming rule is to be
;attached and also modifies the rule-tree appropriately
;calls the procedures go_root and locate
;called from the procedures addrule and update
(DEFUN position (ifs adrul)
 (DO ((rulesearch rules (CDR rulesearch))
      (check 1))
 ((NULL rulesearch) (LIST ifs))
 (SETQ rule_if (CDR (CADAR rulesearch)))
     (COND ((subsetp rule_if ifs)
                (PRINT (SETQ ifs_new (go_root (CAAR rulesearch) ifs)))
                (SETQ result (locate (CAAR rulesearch) ifs_new adrul))
       (return_from result)
           )
     )
 )
)

;Procedure  for locating the root of the rule tree of which rule is  a
;node, recursively moves up using the generality information in a rule
;called from the procedure position and go_root (recursive)
(DEFUN go_root (rule_name ifsr)
  (SETQ gen (CDADDR (CDDDR (ASSOC rule_name rules))))
  (COND ((NULL gen) (SETQ check '2))
        ((EQUAL check 1) (go_root (CAAR gen) ifsr))
        ((EQUAL check 2) (SETQ ifsr (SET-DIFFERENCE ifsr
                  (CDR (CADR (ASSOC rule_name rules))))))
  )
)
```

```
;procedure  for locating the parent of the incoming rule,  this  dives
;down the rule tree till finds the appropriate location for the rule
;called from position and locate (recursive)
(DEFUN locate (rule_name ifsr adrul)
 (SETQ specy (CDADR (CDDDR (ASSOC rule_name rules))))
 (COND ((NULL specy) (LIST ifsr (LIST rule_name) '()))
       (T (DO ((rule_use specy (CDR rule_use)))
      ((NULL rule_use) (LIST ifsr (LIST rule_name) '() ))
      (SETQ rul_if (CDR(CADR(ASSOC(CAAR rule_use) rules))))
              (COND ((SUBSETP rul_if ifsr) (SET-DIFFERENCE ifsr rul_if)
                   (locate (CAAR rule_use) ifsr adrul))
    ((SUBSETP ifsr rul_if)
       (RPLACD (ASSOC 'if (ASSOC(CAAR rule_use) rules))
       (SET-DIFFERENCE rul_if ifsr))
       (RETURN-FROM (LIST ifsr (CADR(ASSOC 'general
                    (ASSOC (CAAR rul_use) rules)))
                    (LIST rule_name)))
    )
  )
       )
)
  )
)

;procedure for reorganizing the knowledge by modifying the rule-tree
;combines two trees to form on rule integrated rule tree
;called from thr procedures addrule and update
(DEFUN fusion (parent sibling)
  (COND ((membert sibling (CDR (ASSOC 'specific
                                      (ASSOC (car parent) rules)))))
        (T (RPLACD (LAST (ASSOC 'specific
                    (ASSOC (CAR parent) rules))) (LIST sibling)))
  )
)
;procedure for reorganizaing the knowledge by modifying the rule-tree
;This simplifies the rule-tree by breaking and constructing
;Called from the procedures addrule and update
(DEFUN fission (parent inter sibling)
  (SETQ parspecy (ASSOC 'specific (ASSOC (CAR parent) rules)))
  (SETQ intspecy (ASSOC 'specific (ASSOC (CAR inter) rules)))
  (IF (membert sibling parspecy)
       (RPLACD parspecy (DELETE sibling (CDR parspecy) EQUAL))
  )
  (IF (NOT (membert sibling intspecy))
      (RPLACD (LAST intspecy) (LIST sibling))
  )
  (IF (NOT (membert inter parspecy))
      (RPLACD (LAST parspecy) (LIST inter))
  )
)
```

```
;main function for pattern matching
;employs functions : pattern-variable pattern-indicator
;     pull-value shove-gr shove-pl
;     restriction-indicator restriction-variable
;called from filter-assertions
(defun match (pattern datum assignments)
   (cond ((AND (NULL pattern) (NULL datum))     ;succeeded
  (COND ((NULL assignments) T)
(T assignments)))
 ((OR (NULL pattern) (NULL datum)) NIL)
 ((OR (EQUAL (CAR pattern) '?)
      (EQUAL (CAR pattern) (CAR datum)))
  (match (CDR pattern) (CDR datum) assignments))
 ((EQUAL (CAR pattern) '+)
  (OR (match (CDR pattern) (CDR datum) assignments)
      (match pattern (CDR datum) assignments)))
 ((ATOM (CAR pattern)) NIL)
 ((EQUAL (pattern-indicator (CAR pattern)) '>)
  (match (CDR pattern) (CDR datum)
 (shove-gr (pattern-variable (CAR pattern))
   (CAR datum)
   assignments)))
 ((EQUAL (pattern-indicator (CAR pattern)) '<)
  (match (CONS (pull-value (pattern-variable (CAR pattern))
                           assignments) (CDR pattern))
 datum
 assignments))
 ((EQUAL (pattern-indicator (CAR pattern)) '+)
  (LET ((new-assignments (shove-pl (pattern-variable
                                                (CAR pattern))
   (CAR datum)
   assignments)))
      (OR (match (CDR pattern) (CDR datum) new-assignments)
   (match pattern (CDR datum) new-assignments))))
 ((AND (EQUAL (pattern-indicator (CAR pattern))
      'restrict)
       (EQUAL (restriction-indicator (CAR pattern)) '?)
            (test (restriction-predicates (CAR pattern)) (CAR datum)))
  (match (CDR pattern) (CDR datum) assignments))))

;procedure to create a new stream of associaton list by matching each
;antecedent with the assertions
;uses : match, add-to-stream
;called from filter-a-list-stream
(DEFUN filter-assertions (pattern initial-a-list)
  (DO((assertions assertions (CDR assertions))
 (a-list-stream (make-empty-stream)))
((NULL assertions) a-list-stream)
   (LET ((new-a-list (match pattern (CAR assertions) initial-a-list)))
   (COND (new-a-list (SETQ a-list-stream
(add-to-stream new-a-list a-list-stream))))))
```

```
;procedure for combining the association list streams received from
;   the procedure filter-assertions.
;uses filter-assertions first-of-stream rest-of-stream combine-streams
;called  from cascade-thru-patterns
(DEFUN filter-a-list-stream (pattern a-list-stream)
  (COND ((empty-stream-p a-list-stream) (make-empty-stream))
(T (combine-streams
          (filter-assertions pattern (first-of-stream a-list-stream))
  (filter-a-list-stream pattern (rest-of-stream a-list-stream))
))))

;procedure for going through all the antecedents
;usesfilter-a-list-stream
;called from use-rule
;feeds the antecedents in reverse order to filter-a-list-stream
;recursively and then revinds to give the required stream
(DEFUN cascade-thru-patterns (patterns a-list-stream)
  (COND ((NULL patterns) a-list-stream)
(T (filter-a-list-stream (CAR patterns)
 (cascade-thru-patterns (CDR patterns)
a-list-stream)))))

;procedure  for  replacing  the  variables  with  their  values  in
;consequents
;usespattern-variable replace-variables
;called from spread-thru-action
(DEFUN replace-variables (conseq a-list)
   (COND ((ATOM conseq) conseq)
 ((EQUAL (CAR conseq) '<)
  (CADR (ASSOC (pattern-variable conseq) a-list)))
 (T (CONS (replace-variables (CAR conseq) a-list)
  (replace-variables (CDR conseq) a-list)))))

;procedure for replacing pattern-variables in the set of conseqs using
;an  assoc  list  from the obtained action stream  tries  to  add  the
;resulting  assertions  to the data and contributes to  a  new  action
;stream
;uses       procedures replace-variables
;called from   procedure feed-to-actions
(DEFUN spread-thru-conseqs (rule-name conseqs cen-list a-list)
   (DO ((conseqs conseqs (CDR conseqs))        ;consider one by one
    (conseq-stream (make-empty-stream)))
       ((NULL conseqs) conseq-stream)       ;termination
     (LET ((conseq (replace-variables (CAR conseqs) a-list)))
 (COND ((APPLY 'OR cen-list) (TERPRI)
                    (PRINT `(RULE ,rule-name says ,@conseq is false)))
       (T (remember conseq)(TERPRI)
                    (PRINT `(RULE ,rule-name says ,@conseq) )
            (SETQ conseq-stream
                                (add-to-stream conseq conseq-stream))
)))))
```

```
;procedure for replacing all pattern-variables using the stream of
;assoc lists passes the consequent stubs along with a assoc list from
the stream to the procedure spread-thru-conseqs
;uses      spread-thru-conseqs
;called from use-rule
(DEFUN feed-to-conseqs (rule-name conseqs censor-list a-list-stream)
  (COND ((empty-stream-p a-list-stream) (make-empty-stream))
(T (combine-streams (spread-thru-conseqs rule-name
conseqs
(first-of-stream censor-list)
(first-of-stream a-list-stream) )
    (feed-to-conseqs rule-name
conseqs
(rest-of-stream censor-list)
(rest-of-stream a-list-stream))))))


;procedure for extracting the decision it calls feed-to-conseqs
handing over the previously obtained stream of assoc list from the
procedure cascade-thru- patterns. The decision stream is empty and
use-rule will return NIL if there are no ways to match the antecedents
to the data or if there are some ways but those ways lead to no new
assertions
;calls      procedures cascade-thru-patterns   feed-to-conseqs
;called from procedure   forward-chain
(DEFUN use-rule (rule a-list)
 (LET* ((rule-name (CAR rule))   ;xtracting rule name
        (ifs (REVERSE (CDR (CADR rule))))
        (thens (CDR (CADDR rule)))
        (unles (CDR (CADDDR rule)))
        (specy (CDADR (CDDDR rule)))
        (gen (CDADDR (CDDDR rule)))
        (a-list-stream (cascade-thru-patterns
                         ifs
 a-list))
        (cen-stream  (feed-to-censors unles a-list-stream))
        (decision-stream (feed-to-conseqs rule-name thens
                                          cen-stream a-list-stream))
 (speci-stream (feed-to-specifs specy cen-stream a-list-stream
                                                  decision-stream))
 ); *
 (NOT (empty-stream-p decision-stream))))

;procedure for forward chaining. steps thru the rule list until it
;find a rule;that produces a new assertion whereupon it starts over at
;the beginning of the rule list. Stops as soon as it fails to find a
;new assertion with any rule in ;the entire list
;calls procedure use-rule

(DEFUN forward-chain ()
   (DO ((rules-to-try rules (CDR rules-to-try))
(progress-made NIL))          cont'd...
```

```
       ((NULL rules-to-try) progress-made)
      (COND ((use-rule (CAR rules-to-try)
                       (add-to-stream NIL (make-empty-stream)))
     (SETQ rules-to-try rules)
     (setq progress-made T)))))

;procedure  for  answering user queries; takes the rule name  and  the
;subject
;associated to answer the queries
(DEFUN ans-query (rule-name subject)
 (COND ((empty-stream-p subject) (PRINT '(give subject name)) )
       (T (LET* ((ex-rule (ASSOC rule-name rules)) ;fetch first
    (mod-rule (APPEND (LIST (CAR ex-rule))
              (LIST (APPEND (LIST 'if)  (LIST (replace-variables-specy
       (CADR (CADR ex-rule))
    (LIST (LIST 'subject subject)) ))
    (CDDR (CADR ex-rule)) ))
    (CDDR ex-rule) ) ) )
     (use-rule mod-rule (LIST (LIST (LIST 'subject subject))))

))))

;procedure for replacing pattern-variables in the set of censors using
;an  assoc  list  and  checking the existence of  the  censor  in  the
;assertions  the  obtained value T or NIL is stored  in  the  censor-
;stream as a list
;uses       procedures replace-variables
;called from   procedure feed-to-censors

(DEFUN spread-thru-censors (censors a-list)
   (DO ((censors censors (CDR censors))        ;consider one by one
(censor-stream (make-empty-stream)))
       ((NULL censors) (LIST censor-stream))       ;termination
     (LET ((censor (replace-variables (CAR censors) a-list)))
        (SETQ censor-stream
(add-to-stream  (membert censor assertions) censor-stream)))))

;procedure  for  replacing all pattern-variables using the  stream  of
;assoc lists passes the censor stubs along with a assoc list from  the
;stream to the
;procedure spread-thru-censors
;uses      spread-thru-censors
;called from  use-rule
(DEFUN feed-to-censors (censors a-list-stream)
  (COND ((empty-stream-p a-list-stream) (make-empty-stream)) ;nil
(T (combine-streams (spread-thru-censors
       censors
(first-of-stream a-list-stream))
    (feed-to-censors
 censors
(rest-of-stream a-list-stream))))))
```

```
;procedure for replacing the variables with their values in
;specificity rules or when some query is asked by the user
;usespattern-variable replace-variables-specy
;called from feed-to-specifs
(DEFUN replace-variables-specy (specy-if a-list)
   (COND ((ATOM specy-if) specy-if)
     ((EQUAL (CAR specy-if) '>)
(COND ((EQUAL (pattern-variable specy-if) 'subject)
          (CADR (ASSOC (pattern-variable specy-if) a-list)))
  (T specy-if)))
 (T  (CONS (replace-variables-specy (CAR specy-if) a-list)
   (replace-variables-specy (CDR specy-if) a-list)))))

;procedure for eliminating those elements from the assoc-list which do
not give positive decision due to censors by checking those elements
of censlist which yield value T
;uses first-of-stream rest-of-stream make-empty-stream combine-
streams eliminate-cens
;called from feed-to-specifs
(DEFUN eliminate-cens (cens-list a-list)
  (COND ((empty-stream-p a-list) (make-empty-stream))
    (T (combine-streams (COND ((APPLY 'OR (first-of-stream cens-list))
          (make-empty-stream));if cens remov
    (T (list (first-of-stream a-list))))
      (eliminate-cens (rest-of-stream cens-list)
                      (rest-of-stream a-list)) )) ));recur

;procedure for manipulating the specificity part of the active-rule
;takes the indicated parameters, checks for empty decilist, eliminates
;those elements from a-list which are unnecessary(censors) then there
;are two loops for each of the elements in a-list and each of the rule
;in the specificity part of the parent rule
;uses empty-stream-p make-empty-stream eliminate-cens replace-
;        variables-specy
;called from use-rule
(DEFUN feed-to-specifs (specy cens-list a-list deci-list)
  (COND ((empty-stream-p deci-list) make-empty-stream)
(T ((SETQ a-list-new (eliminate-cens cens-list a-list))
    (DO ((a-list1 a-list-new (CDR a-list1))
 (specy-list (make-empty-stream)))
 ((NULL a-list1) specy-list)
 (DO ((rule1 specy  (CDR rule1)))
                    ((NULL rule1) specy-list)
     (LET* ((ex-rule (ASSOC (CAAR rule1) rules))
    (mod-rule (APPEND (LIST (CAR ex-rule))
            (LIST (APPEND (LIST 'if)
                                (LIST (replace-variables-specy
        (CADR (CADR ex-rule))(CAR a-list1)))
              (CDDR (CADR ex-rule))))(CDDR ex-rule))))
                (SETQ specy-list (APPEND specy-list (use-rule
           mod-rule (LIST (CAR a-list1)) ) ))
)))))));calling use-rule with specificity rule
```

CD	Change directory
CHKDSK/F	Check status of files on disk
COPY	Copy files
DEL	Delete files
DIR	Directory of files
DISKCOPY	Copy a diskette
FORMAT	Format a diskette
MD	Make a directory
REN	Rename a file
RD	Remove a directory
TYPE	Display contents of a file

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

[2] Winston, P.H. — Artificial Intelligence Addison-wesley (1984)

[3] McCarthy, J. — Circumscription - A form of nonmonotonic reasoning Artificial Intelligence 13 (1980) 27-39

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

[5] McCarthy, J. — "Programs with common sense," in semantic information Processing MIT Press, Cambridge Mass. (1968)

[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, MSc thesis, Department of Computer Science, University of Illinoi Urbana, IL. (1985)

[8] Reinke, R. Michalski, R.S. — Incremental learning of concepts description, 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)

[10] Winston, P.H. — LISP Addison-wesly. (1984)

[11] Steel, G. Jr. — Common Lisp : The language DEC, USA.

[12] Bibel, W. Jorrand, Ph. — Fundamental of Artificial Intelligence Springer Verlag. (1986)

[13] Elaine Rich — Artificial Intelligence McGraw Hill (1983)

[14] Michalski Carbonell Mitchell — Machine Learning Springer-verlag (1985)

[15] Leonard Bolc (E.d.) — Computational models of learning Springer-verlag (1987)

[16] Bratko, I. Lavrac, V. (E.d.) — Progress in machine learning Sigma Press, Sussex, England (1987)

[17] Stuart Savory (E.d.) — Artificial Intelligence and Expert Systems John wiley & Sons

[18] Bahrami, A. — Designing Artificial Intelligenc based software, John wiley & Sons and Sigma Press (1988)

[19] Jain N. K. — Variable Precision Logic : Heirarchical Censored Production Rules System, M.Tech Dissertation, School of Computer and Systems Sciences Jawahrlal Nehru University New Delhi. (1989)

[20] Bharadwaj, K. K. Jain N. K. — "Hierarchical Censored Production Rules (HCPRs) System" (under revision : Journal of Data & Knowledge Engineering)