708

A RULE—BASED EXPERT SYSTEM IN THE DOMAIN OF LAW — AN ANALYSIS

733

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

Sup: Saxena, P.C.

Notes
1. Theses (M.Tech) - Jawaharlal Nehru University, 1988
2. Typescript

ASHUTOSH CHATURVEDI

38p + reference

SCHOOL OF COMPUTER & SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI—110067
INDIA
1988

A RULE-BASED EXPERT SYSTEM IN THE DOMAIN OF LAW - AN ANALYSIS

PREFACE

The research work embodied in this dissertation has been carried out in the school of computer and Systems Sciences , Jawaharlal Nehru University, New Delhi. The work original and has not been submitted so far, in part or full, for any other degree or diploma of any university.

[illegible]
4/1/89
ASHUTOSH CHATURVEDI

[illegible]
DR. P.C. SAXENA
SUPERVISOR

[illegible]
PROF. KARMESHU
DEAN

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

CONTENTS

ACKNOWLEDGEMENTS

I wish to express my gratitude to Dr. P.C. Saxena Associate Professor, School of Computer and Systems Sciences, J.N.U., New Delhi for his motivation, guidance and help which led to the culmination of this work.

As is widely known, the expertise of an expert is an indispensable component of any expert systems work. In this respect, I am very thankful to Mr. Arvind Kumar, a noted lawyer of Delhi for his invaluable help in providing literature and guidance.

ASHUTOSH CHATURVEDI
M.TECH (SC & SS)
J.N.U., NEW DELHI

Chapter 1

THE IMPORTANCE OF EXPERT SYSTEMS

Intelligence, as it has been described in dictionary is the capacity to apprehend facts and propositions and their relations and to reason about them. The credit for the origin of Artificial Intelligence as it is known now must be given to M. Turing who gave the concept of Stored Program Computer which was different from the earlier computers in the sense that the earlier computers were actually dedicated machines that had to be wired differently to solve different problems. But Artificial Intelligence in its current form began around 1966 with the creation of LISP, the first artificial intelligence language. Since then, many systems have been designed in the field of artificial intelligence or AI as it is called such as the chess-playing computer, computerised mathematical proofs, the famous psychoanalysis program ELIZA etc. By now, AI has become divided into different specialised areas of study such as : natural language processing, knowledge representation, learning, Expert systems etc.

An Expert System is a program that contains knowledge about a certain field, and when interrogated, responds in a way, that a human expert would respond.

There are various advantages inherent in the concept of making commercially viable expert systems.

The first advantage is 'availability'. Since Many expert systems can be created, whereas there may be a limited number of human experts, making it virtually impossible in many situations to have an expert available when needed.

The second advantage of an expert system over human experts is its 'continued peak performance' whereas the reliability of a human expert is sensitive to changes in his mental state.

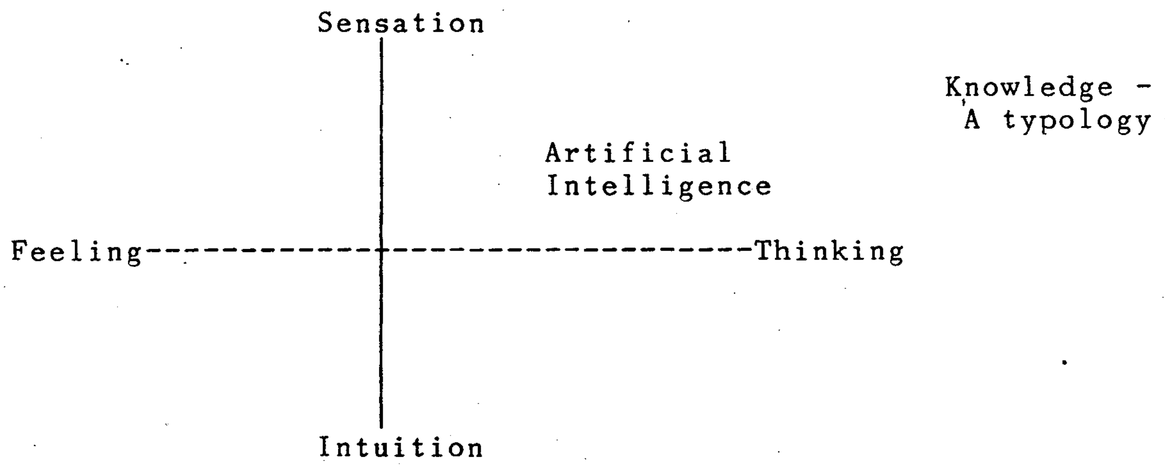
An expert system is an 'impersonal tool' and so can be more effective than a human expert who may have personal likes or dislikes.

MYCIN, designed to help physicians diagnose certain bacterial diseases was world's first successful expert system. Another example of a commercially viable expert system is PROSPECTOR, an expert in geology. Since early 1980s, various dedicated expert systems in fields like computer configurations, Robotics, tax consulting, insurance advice, legal aid etc. have been written. It is believed that in the years to come there will be a large market for expert systems for "personal" use such as at offices and homes.

1.1 Rules and Knowledge

Most of the useful work that is being done in AI today is heavily based upon the collection and usage of heuristics, condition-action rules etc., because they can be used to exhibit something like intelligence.

The famous psychiatrist Carl Jung has given a method of looking at types of human experience, and the ways that we process that experience.



This simple presentation shows parts of opposing functions which are available to process our experiences: sensation and intuition for perceiving experience and thinking and feeling for evaluating experience.

We perceive events in the physical world by our sensation function, which is implemented by our five physical senses. These sensory perceptions are passed on to the thinking function which organises the sensory input and hands that information to a feeling function. Finally, the intuitive function is a kind of inner perceiving function which extends past experience, allowing us to look beyond our present situation. In fact, the capabilities of AI and of all the computer science are limited to the perception of physical inputs and their processing by logical-deductive thinking processes. We can also emulate feeling and intuitive functions on our computers by discovering and codifying non-logical expert activities using our logical deductive tools.

1.2 What constitutes an Expert System

Every expert system has 2 parts : the knowledge base and the inference engine.

1.2.1 Knowledge base

It is a database that holds specific information and rules about a certain subject. The two important terms are:

Object :	The conclusion that is defined by its associated rules.
Attribute:	A specific quality that together with its rule, helps define the object.

A knowledge base is a list of objects with their associated rules and attributes. Normally, an object is defined by a list of attributes which the object either 'does' or 'does not' possess. For example, an expert system that identifies various types of fruit may have a knowledge base like this :

Orange

> has been grown on tree
> has round shape
> has not been grown in the north
> has orange colour

Apple.

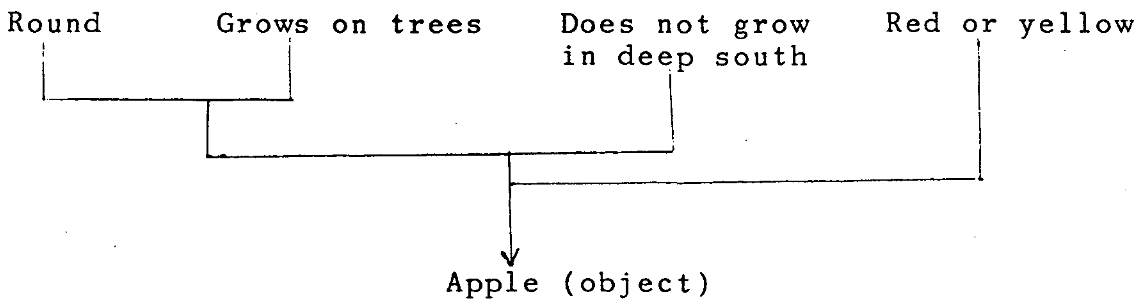
1.22 Inference Engine : Once the knowledge base consisting of rules has been formed , the next important taksk is to chain through the rules to reach a conclusion, which involves essentially the following :

1. Identify the rule as relevent to the conditions (premises) of the problem situation. This step may actually result in finding more than one rule from the knowledge base. If so,
2. We must resolve the conflict among competing rules, the result being the selection of one rule.
3. We can now excute the rule i.e. we can reach the conclusion implied by its premises.

There are two broad categories of inference engines : 'deterministic' and probabilistic. A deterministic expert system would give a definite answer to most queries. Whereas a probablistic expert system can give an answer which can be treated as only probable or having a certain success ratio. The basic ways to construct the inference engine are three : forward-chaining, backward-chaining, and rule value.

1.221 : Forward - Chaining Method : It is sometimes called data - driven. Here inference engine acts on the information provided by the user to move through a network of logical ANDs and ORs until it reaches a terminal point, which is the object. Thus, a forward chanining inference engine starts with some information and tries to find an object that fits the information.

For example in the diagram given below when the forward chaning inference engine is given the proper attributes it arrives at the object apple.

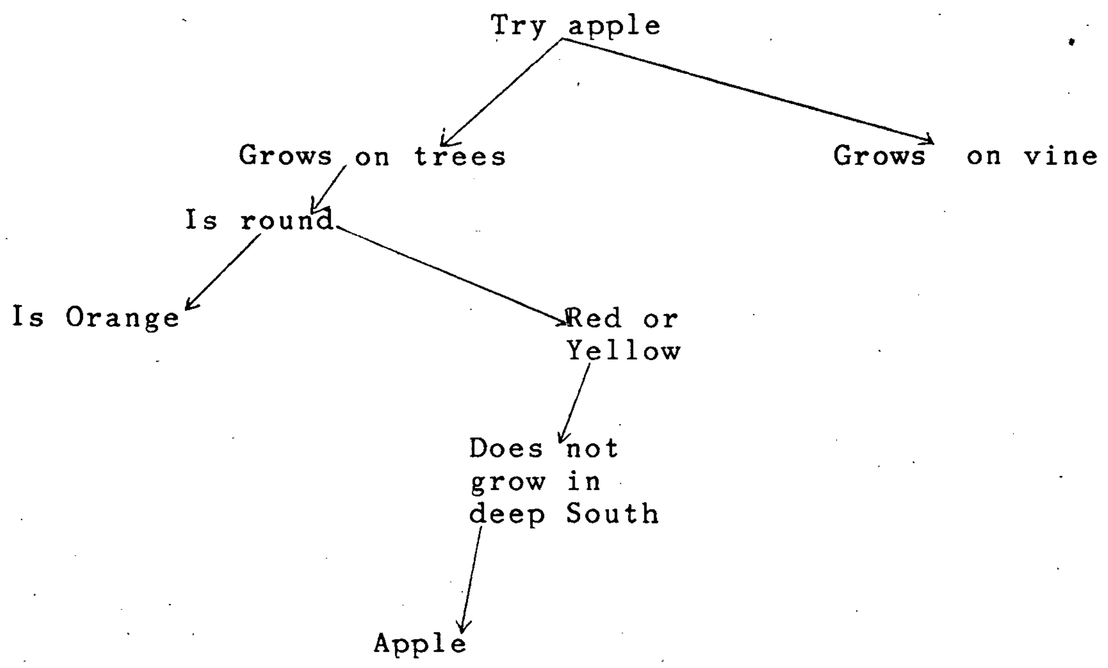


(Forward chaining to the object apple)

1.222 **The Backward Chaining Method**

In this case, the inference engine starts with a hypothesis (an object) and requests information to confirm or deny it.

For the earlier example of a fruit, a backward chaining Inference engine would take the apple as an object and then would attempt to verify it by checking for its attributes.



Thus backward-chaining prunes a tree, the opposite process of building a tree which happens in case of forward chaining.

1.223 **The Rule Value Method**

In this method the general theory is that the system makes request for such information as will remove the most uncertainty from the system. Thus at each stage, the question which is selected is one that makes the most rapid progress to a conclusion. The rule-value systems are difficult to implement because the real-life knowledge bases are so large that the system can not decide as to which questions will take it closest to the goal.

1.3 Expert Systems in a specific problem area - Law

The field of law can lend itself to treatment by Expert Systems technology in a very profitable manner. Because of the large number of rules that legal subjects generally have, it becomes a trying task for a lawyer to go through the voluminous rules manually and also be able to interpret them in the precise manner. Thus having an expert system to store and retrieve rules for a given problem area would greatly facilitate the tasks of lawyers, academicians, business houses and others concerned with the subject. A large number of expert systems in the law field have been implemented. Extensive discussions with lawyers, resulted in the identification of an area for developing an expert system.

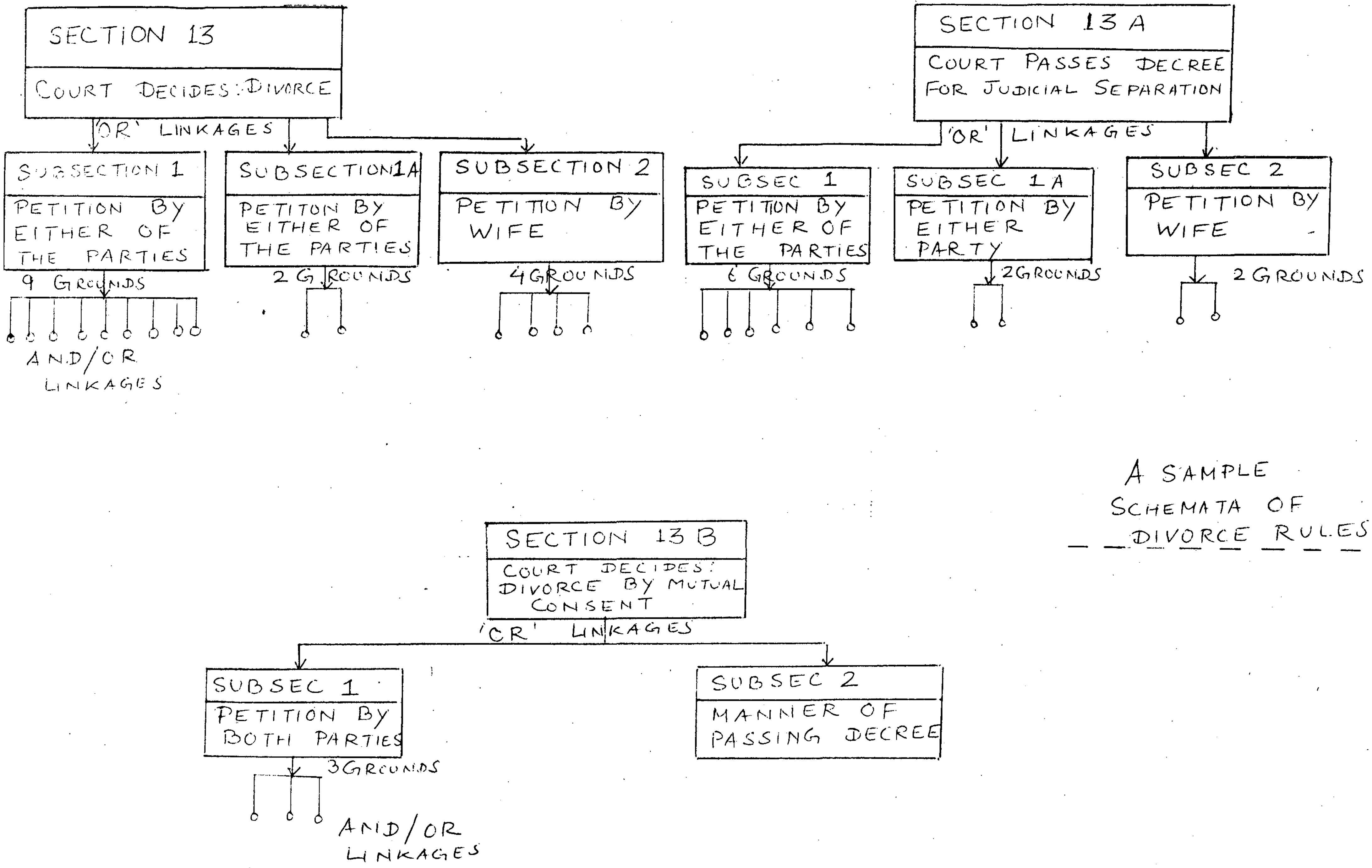
The example taken in this dissertation also relates to the legal field. An attempt has been made to show as to how the expert system can be used in the field of law, which is a relatively newer concept.

The subdomain of the problem which has been taken out of the vast domain of legal problems deals with the Hindu Marriage Act, 1955. This Act extends to the whole of India except J&K, and also applies to Hindus domiciled in territories to which this act extends who are outside the said territories (the act also applies to the Hindus domiciled in India but who are living outside India).

The sections which deal with divorce, judicial separation and divorce by mutual consent are sections 13, 13A, and 13B respectively.

A block diagram depicting the various sections, sub-sections and various grounds in which they can be invoked is given below (next page)

Thus it can be seen that there are sections dealing with divorce problems, each section having a number of subsections, and each subsection has a number of grounds. Some of these subsections and grounds are common. Furthermore, there are AND/OR linkages between the applicability of various grounds. Thus, this problem presents a good case for treatment by expert system technology. Discussions with legal expert together with going through the voluminous summary of sections, sub-sections and ground rules in all their complexity helped a



A SAMPLE SCHEMATA OF DIVORCE RULES

lot in building up a framework of a prototype of an expert system which is presented in a greater detail in the next chapter.

Chapter 2

DESCRIPTION & ANALYSIS OF AN EXPERT SYSTEM

Although, the two fields are quite far apart, yet a parallel can be drawn between the problem at hand and that of constructing a medical diagnostic system. When a doctor has to diagnose a patient, he has a list of probable diseases in mind. Each of these diseases has a set of symptoms as its indication, so the doctor looks for these symptoms in the patient. If the symptoms for a particular disease are not present then the patient is examined for the next disease and this process goes on until the correct disease is found out. The "patient" in the given problem is the client of a lawyer, i.e. a husband or wife or both. The "diseases" are the number of options of settlement before the spouses such as divorce, judicial separation etc. The "symptoms" here are the various sections, subsections and ground-rules as established in the Hindu Marriage Act.

Thus, what is needed in the formulation of an expert system for the solution of the given problem is the following facilities at the highest level :

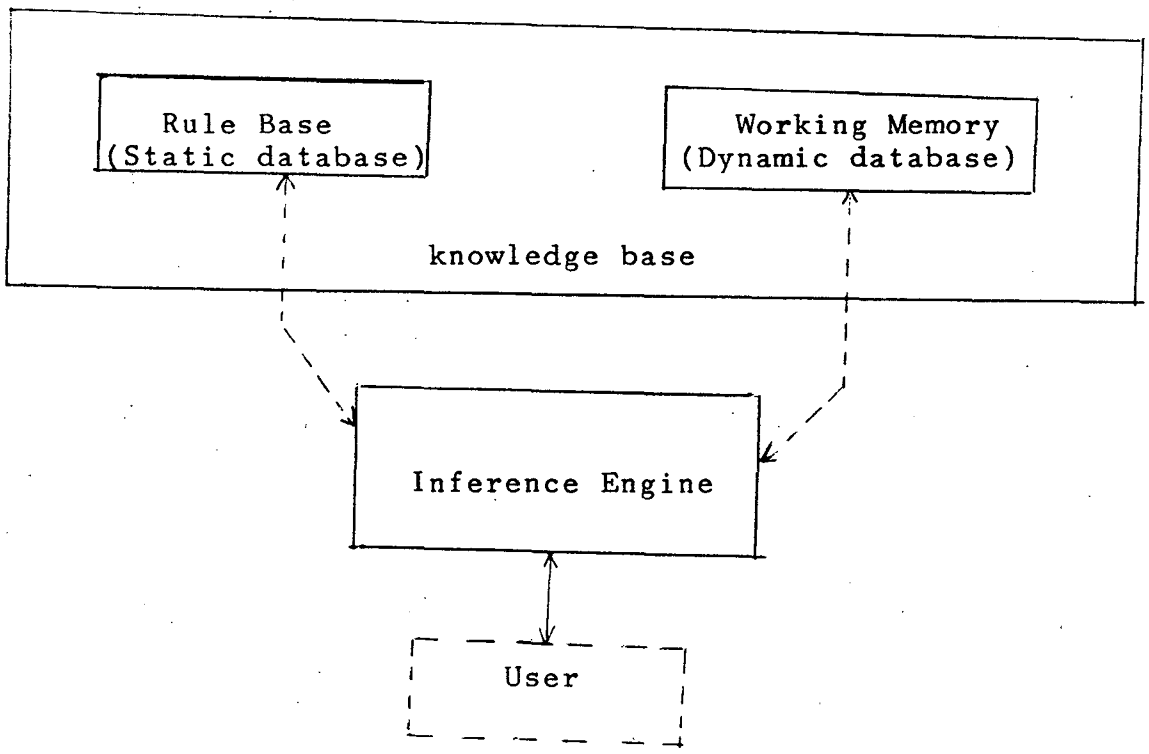
(1) A facility to store a large number of rules.

(2) A facility to map these rules onto real-world, i.e. a

facility to check the applicability of some or all of these rules for a given physical entity.

Two kinds of storage are needed in this problem. One is the collection of facts and rules which does not change during program execution. This is called static database. The second type of storage needed consists of facts obtained from the user during the consultation process that apply only to the current consultation.

Thus the following type of structure is needed :



Such a structure is called a production system which is a common type of expert system. Using this model in mind, a primitive expert system has been developed here. This system was developed using an IBM Personal Computer and the only software that has been used here is the Turbo Prolog compiler.

2.1 Use of Language PROLOG

The language that has been used here is Prolog, an AI language (more precisely Turbo-Prolog), which is different from the conventional computer languages in the sense that it is an object-oriented language. The conventional languages are procedure-oriented languages which distinguish between a program and the data it uses. Whereas Prolog uses only data about objects and their relationships. A brief note about Prolog would be in order before we start a detailed discussion about the program.

As we have said, Prolog is an object oriented language, which means that the collection of facts and rules (the so-called data of conventional languages) and the relationships between objects, the control structure and the user interface (the so called program of the conventional languages are both written in the language Prolog. In fact, Prolog is its own inference engine, which performs unification (i.e. matching), decides the order in which to scan the rules, and performs conflict resolution. Prolog uses backward chaining, i.e. it starts at the goals and works backwards. Moreover, Prolog is limited to depth-first scanning. All rules concerned with a particular goal are scanned as deeply as possible for a solution before Prolog backtracks and tried to prove another goal. These concepts will be clearer, once we come to a detailed analysis of the system.

The given law Problem requires the system to have the ability of testing through all the given conditions, and in the event of any condition failing, going backwards to have another variable binding and then continue to test the conditions. An important feature of Prolog execution called backtracking does just this. In this, the solution of a compound goal proceeds from left to right, if any condition in the chain fails, Prolog backtracks to the previous condition, tries to prove it again with another variable binding, and then moves forward again to see if the failed condition will succeed with the new binding.

Prolog uses a database of 'facts' and 'rules'. Facts are the assertions of something which is true expressed in an encoded form. Facts are expressed in association with 'objects', which represent an entity or a property of an entity in the real world. An associated term is 'Relation' whyich is a name that defines the way in which a collection of objects (or objects and variables referring to objects) belong together. For example, consider,

Has-a(X,X)

It represents a fact that an entity X 'has a' another entity X. Here, 'Has-a' is a relation. The entire expression before the period is called a 'predicate'. Predicates are functions with values true or false and they express a property or a relationship. A 'rule' is an expression that indicates that the truth of a particular fact depends on one or more other facts.

Every rule has a 'conclusion' (or head) and an 'antecedent' (or body). The antecedent consists of one or more 'premises'. The premises form a disjunction or conjunction of goals (as decided in the rule) and accordingly all or some of them must be satisfied for the conclusion to be true. A comma expresses an 'and' relationship and semicolon expresses an 'or' relationship. Rules can use both variables and constants as their arguments.

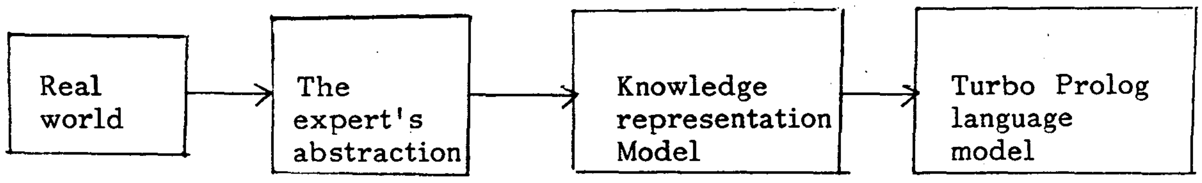
Prolog executes using a matching process. The process by which Prolog tries to match a term (a term is a simple object, variable or structure) against the facts or heads, of other rules in an effort to prove a goal is called 'unification'. Predicates unify with each other if

- They have the same relation name
- They have the same number of arguments
- All argument pairs unify with each other.

Unification is similar to parameter passing in procedural programming. Values for one term are passed to another term, binding any variables in that term.

2.2 Implementation Details

The following flow diagram shows how Prolog is applied to the real world.



A Turbo Prolog program consists of two or more sections. The main body of the program, the 'clauses section' contains the clauses and consists of facts and rules. The relations used in the clauses of the clauses section are defined in the "predicates section". The "domains section" defines the type of each object. There are various domain types in Turbo Prolog such as charts, symbol, real, string etc.

The subsequent discussion is based on the program printout attached near the end of this report. The domains section here (kindly refer to the program), contains the types of various objects. The variable 'descn' refers to the various types of decisions possible in the court in the divorce cases, such as divorce, judicial separation, mutual consent divorce, etc. The variable 'grnds' refers to the various grounds whose fulfilment leads to some decision or the other. Likewise, the variable 'query' which if of string type is used to ask questions by the system. The variable answer is used to input the answers of user to the queries made by the system.

The domain section is followed by database section. The two database predicates defined here are 'xpositive' (grnds)' and 'xnegative (grnds)'. These together with the built in predicates 'asserta (fact) and 'retract-(fact)" help in creating and retrieving a dynamic database of facts. How actually this is done will be explained a little later.

In the diagram given in the last chapter, it has been made clear that sections 13, 13A and 13B deal with divorce cases. Each of this sections consist of a number of subsections. Any of the subsections can be invoked for the decision in that particular section to be made. Thus there are OR linkages between the subsections within a section. Furthermore, for each subsection to be invoked a number of conditions (either all or some of them) have to be satisfied. Thus there are AND OR linkages between the conditions within a subsection. Some of these conditions require the fulfillment of multiple subconditions before they are themselves fulfilled. Also some of these conditions require just one of the many subsections for them to be true. Thus we can see that there are AND/OR linkages within a particular condition between different subconditions.

For example, for divorce, in the broadest manner possible the following structure of rules is present.

```
"Divorce is admissible IF
   (either of the spouses presents a petition
   AND conditions for Section 1 are fulfilled)
OR (either of the spouses presents a petition)
   AND conditions for Section 1A are fulfilled)
OR (the wife presents a petition
   AND conditions for Section 2 are fulfilled)"
```

Similar constructs are present for the other decisions such as decree for separation by mutual consent.

In a greater level of detail if we view the problem, then we will see that Section 1 can be invoked in the following cases which becoome clear by inspecting the following rule :

```
"Invoke Section i  IF,
   Conditions for CONVERSION are present
OR IF
   Conditions for RENUNCIATION OF WORLD are present
OR IF
   Conditions for NOT HEARD LIVING are present
OR IF
```

The conditions of say CONVERSION can be viewed in a greater level of detail.

"A RELIGIOUS CONVERSION" is said to happen IF, the respondent abdicates his religion by an act of remuneration

AND

The respondent adopts another religion by formal conversion".

All the judicial decisions are treated as hypotheses some or all of them may be possible. The conditions and subconditions discussed above are given as premises of these hypotheses. Then these premises are themselves set up as hypotheses, with their subcontinued as their premises.

The total program can be divided into three parts.

(A) Declaratory and query management part

(B) knowledge-base (rules) part

(C) User Interface part

For initiating the expert system, the user has to type in a goal as "go". Then, following interchanges take place between the various parts of the program.

The queries which are asked are described below in a brief manner.

First of all the system asks,
"What do you want?"

At the same time it lists out a list of options within which to choose.

Now the user can say,
"I want to know whether divorce is possible"

or

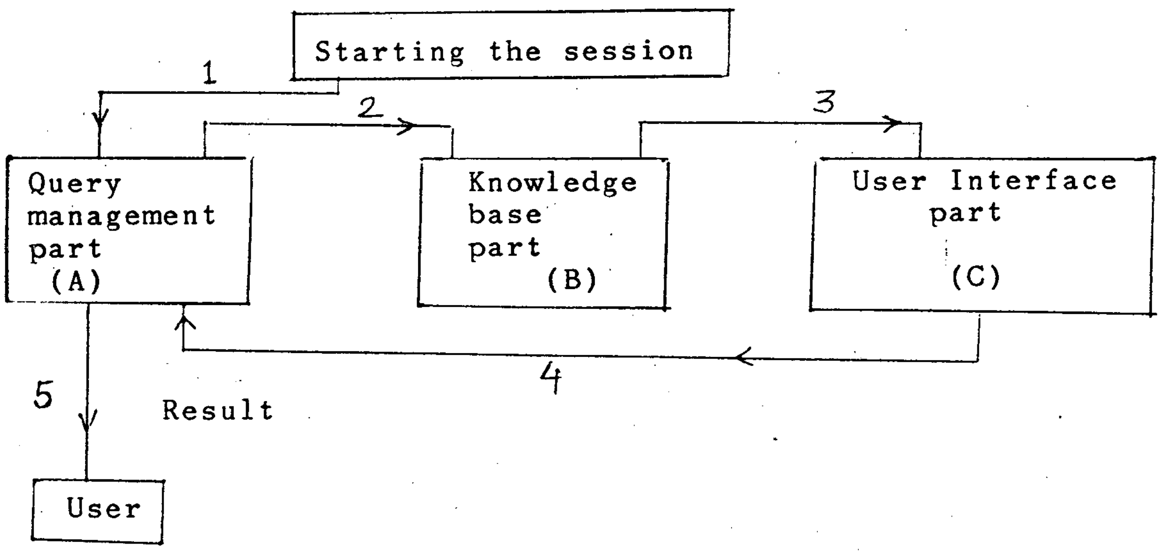
"I want to know whether judicial seperation is possible"

or

"I want to know whether divorce by mutual consent is possible".

Depending on these, the system goes ahead and checks the truthfulness of the appropriate hypothesis. Then depending on te hypothesis, it checks the truthfulness of appropriate conditions, and subconditions by asking the user various questions, which will be clear from the program listing in the appendix attached at the end.

2.3 System Flow Pattern



Index of Execution Flow

(1) User specifies the goal 'go' which indicates the query management part

(2) A checks for a particular hypothesis in B.

(3) B goes deeper inside the rules to get at the most elementary predicate and triggers C

(4) C checks for the concerned data base predicate in A.

Depending upon whether this database predicate in true or not, a new set of execution flows occur in the order 2, 3 & 4.

At the end, the result is communicated to the user through flow No.(5).

How exactly this flow occurs would now be discussed in detail.

After data base section comes the predicates section which declares all the predicators to be used. These predicates may be true or false. The symbols within brackets are the arguments of the predicates, which are themselves defined in the domains section. The same predicate may be true or false depending on the values of the arguments.

The actual program starts from the clauses section. As user types in 'go', the 'go' predicate is tested and the predicates which are its premises are tested. The symbol ',' is for conjunction. The predicate xpositive (grnds) is used to store facts proven true and xnegative (grnds) is used to store facts proven false. First of all the dynamic databases are initialised by clear facts predicate. Then the hypothesis (decsn) is tested. The sign '1' is a predicate which is built in to always succeed. It is called a cut. Then the Prolog looks for the first instance of hypothesis (Decsn) which it finds to be 'Hypothesis (divor)'. Then it checks for various premises of this Hypothesis which are further described in the knowledge base. So it first comes to test the predicate "method (eithpetn)". As we have said earlier, the result of an IF statement is separated from its premises by a ':-' sign. So the premise of 'method (eithpetn)' says, "positive" ("has the petition been moved by either of the spouses (Y/N)?", eithpetn)"

This positive clause now looks for a match all through the program from the start. So it tries to find a match with the first positive clause, namely "positive (-, grnds)". As we know, Prolog executes by a process of unification. So it ignores the argument corresponding to '-' which happens to be the query and binds 'Grnds' to 'eithpetn'. Then the predicate xpositive (eithpetn) is tested. This will fail since initially all database predicates are empty. The progrm then backtracks to the second 'positive (query, grnds)' predicate./ Since xnegative (eithpetn)' is currently empty so together with 'not' it succeeds and we come to the 'ask' predicate. Here the query earlier mentioned in user-interface part, is put to the user. When the user gives answer, the system remembers it as 'yes' by making 'xpositive (grnds)' as true or 'no' by making 'xbegative (grnds)' as true. This is done by the built in 'assertafact) predicate. So if by this process, the predicate method (eithpetn) comes out to be true, then the system tests for the next premise, namely, 'sec(one)' Its testing requires the testing of another ruele with a number of premises. If assuming that all predicates that are ANDed are true then the hypothesis becomes true which is reflected by system declaring the parameter of that hypothesis as the decision. But if suppose any of the premises fails, leading to the failure of the hypothesis. In that case, the system goes on to the testing of the next hypothesis with another binding namely hypothesis (judsep). It again restarts the same process, going through the same manner of execution flow and of course asking the different questions. The cut performs the heuristic function of reducing the search space, once a 'decision' matching the 'grnds' is found, the search is terminated.

The various levels of conditions and subconbditions makes about 35 rules at the lowest level. Thus so many queries are stored in the user interface.

Since Prolog looks for a matching in a sequential manner, so the ordering of hypothesis and premises is very important.

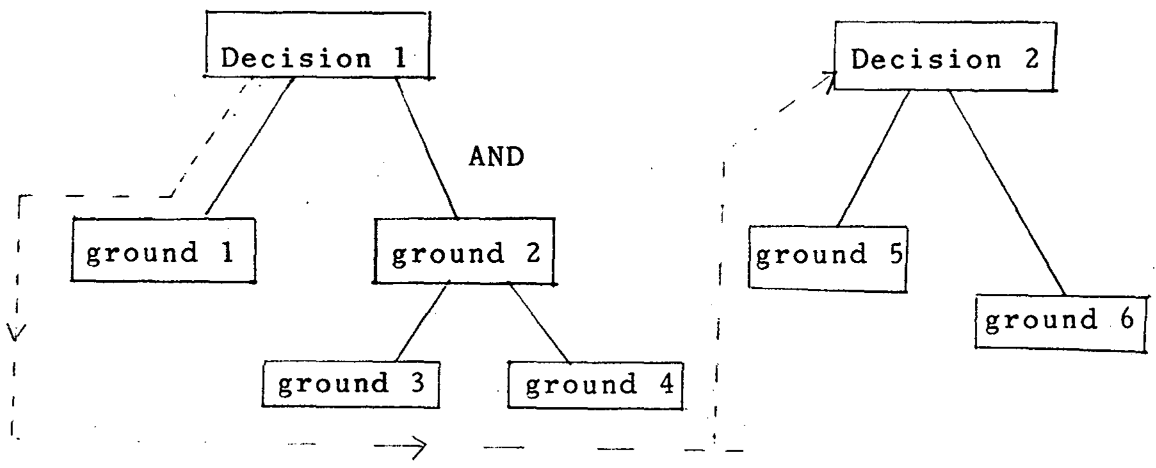
Also, it is important to know that clauses of same predicate are to be grouped together. The 'method' predicate is used to store the method of presenting petitions i.e. single, jointly etc. Likewise, there are predicates such as "condtn (grnds)", 'subsec(grnds)" etc. which (each of them) require a large number of premises.

Two important features of this system will become clear by seeing its operation or by looking at the program deep enough.

The first is that the system prunes the search space by not asking the questions which have been asked earlier.

This is achieved through the device of dynamic data bases. The query which the system asks is given an answer in 'Y" or 'NO' and the system remembers this answer by the remember predicate which stores this answer in 'xpositive' if it is 'Y' and if 'N' then it is stored in 'xnegative'. This decides the fact whether the question will be asked next time or not. Thus the user is spared the botheration of getting asked the same question again and againb. Thus it is a very powerful feature.

The second feature is that the system does not ask the subsequent subqueries if the earlier subqueries are answered in 'no' and if the subqueries are linked by an 'AND'. This is achieved by the cut. Thus the search space is pruned by this method. Let's illustrate this by a diagram.



Let's assume that ground 1 fails. Then the control will try and test the hypothesis clause with another variable binding say 'decision 2', instead of fruitlessly searching ground 2, ground 3 etc. of hypothesis 1. Thus there is considerable saving in terms of processing time. One part of the search tree is completely written off momentarily for one consultation session.

2.4 The Notion of Certainty

The examples of rules which have been considered might give an impression that the rules, as stated, and the interaction with the user, as described are relatively well defined. This is not a typical case.

In general, experts' knowledge has some kind of certainty associated with it, as is the case with the input from the user at decision time. The important questions with regard to certainty of knowledge are : How do we decide how certain a particular piece of knowledge is a priori? and if the premises themselves are uncertain, how do we combine them to find the conclusion of a rule with a reasonable cer0-tainty..

Every programming language has some explicit or implicit equivalent of the following statement :

IF some things are true
THEN some other things are true
ELSE yet other things are true

But there is a very basic difference between the traditional IF-THEN rules and the production rules used in knowledge based systems, such as certainty, completeness and dynamism. If the rules are of the type in which the conclusions are completely determined by the truth of the conditions i.e. premises they they are called 'deterministic' rules. Such rules occur generally in language such as COBOL where for example if the input transaction is Add-a-Record, then the Add-a-Record process should definately be executed.

The knowledge-based system rules are different in the sense that when we make an expert rule, we may not be absolutely certain that either the premises or the conclusions are true.

For example, for the expert system being discussed in this report, consider one of the rules :

"Judicial separation is admissible
IF Petition is presented by either of the spouses
AND OR The respondent was incurably of unsound mind
OR the respondent was suffering from incurable and virulent leprosy
OR the respondent was suffering from veneral disease)"

Now several questions come to one's mind after inspecting this seemingly simple rule. Each of the premises given is amenable to a closer scrutiny in the following manner:

¶

Cruelty	What was the type and extent of cruelty?
	What was the effect of cruelty?
Unsound mind	Was the person incurable ? was the person continually afflicted with mental illness?
	What was the extent of mental illness?
Veneral disease	What is the extent of veneral disease? What parts of the body does it effect?
	Is it of communicative form?

It would appear that having a large number of conditions would increase the certainty of the rules. In fact this is what has been actually done in the system implemented as such. For each of the grounds admissible queries have been framed at a finer level of detail. But still the fact can not be discounted that each of the new factors may be true only to a certain degree and certainly that degree varies from person to person.

In our expert system, each condition at the most elementary level has a degree of uncertainty associated with it. Thus we can associate a probability P(gi) for each elementary condition gi. There are various elementary conditions' g1, g2, g3 .... gn

And the final decisions can be given the probabilities

P(d1), P(d2), ... P(dn)

i.e. if the decisions are d1, d2,........... dn.

Thus the uncertainty of premises can be transmitted to the uncertainty of the conclusion and this uncertainty itself can be computed.

Suppose the following relations exists between the decision and the conditions

"Invoke d_k IF

g1 AND g2 AND .... AND gm"

Thus the uncertainty associated with d_k is given by :

$P(d_k)$ = P (g1 AND g2 AND .... AND gn)

Also, if we assume that the fulfillment of a particular condition is independent of that of any other condition then from law of independent probability, we get

$P(d_k) = P(g1) * P(g2) * \ldots\ldots\ldots\ldots * P(gn)$

If we consider the conditional probability,

$P(d_k/g1$ AND g2 AND .... gm) = 1

This is because the decision d_k has as its preconditions, the conditions g1, g2 ..... gm and if all of them are true, then the decision d_k is certainly to be taken, but because of the uncertainty, it can have a value different from 1./

Now consider another conditional probability which is an apriori combinational conditional probability.

P(g1 and g2 AND .... AND gm/d_k) ......(A)

According to laws of conditional probability, it can be written as :

$$\frac{P(g1\ AND\ g2\ AND\ \ldots\ldots\ldots\ gm)\ P(d|g1\ AND\ \ldots\ldots gm)}{P\ (d_k)}$$

Due to reasons discussed above, the above can be reduced to,

$$\frac{P(g1)*P(g2)*\ \ldots\ldots\ldots *P(gm)*(\text{Uncertainty Factor})}{P\ (d_k)} \qquad (B)$$

Now, (A) can be further reduced to the following form using independence of conditional probability.

$$P(g1|d_k) * P\ (g2)|d_k) * \ldots\ldots\ldots P(gm|d_k) \qquad (C)$$

Since (A) & (B) were equivalent, hence (B) & (C) are equivalent. Now (B) is of a form which can be computed if we have the probabilities of fulfillment of individual conditions. These can be known from the expert of the field based on his/her knowledge of frequency of occurrences of these conditions. This can lead to computation of $P(d_k)$ and thence of (B). Thus we can get (C). If we apply the approximation

$$P(g1/d_k) = P(g2/d_k) = \dots\dots\dots P(gm/d_k)$$

Then we can get the independent conditional probabilities $P(g1/d_k)$ by computing the mth root of (C).

In the case of conditions linked with an 'OR', if we consider the conditions to be mutually exclusive then we can say,

$$P(g1 \text{ OR } g2 \text{ OR } \dots\dots\dots \text{ OR } gm/d_k) = P(g1/d_k) + P(g2/d_k) + \dots\dots\dots\dots + P(gm/d_k) \qquad \text{(D)}$$

This is because

$$P(g1 \text{ OR } g2 \text{ OR } \dots\dots\dots \text{ OR } gm) = P(g1)+P(g2)+\dots\dots+(gm)$$

Also from laws of conditional probability

$$P(g1 \text{ OR } \dots\dots \text{OR } gm/d_k) =$$

$$= \frac{P(g1 \text{ OR } \dots\dots\dots \text{ OR } gm)*P(d_k/g1 \text{ OR } \dots\dots \text{OR } gm)}{P(d_k)}$$

Here, $P(d_k/g1 \text{ OR } \dots\dots\dots \text{ OR } gm)$ = uncertainty factor, because these are the preconditions for d_k.

So, $P(g1 \text{ OR } \ldots\ldots\ldots\ldots \text{ OR } gm/d_k)$

$$= \frac{P(g1) + P(g2) + \ldots\ldots\ldots\ldots P(gm)}{P(d_k)} * \text{(Uncertainty Factor)} \qquad \text{(E)}$$

Since we can find (E), from (D) and (E), we can find $P(g1/d_k)$ in the same manner as discussed earlier for the 'AND' case, if we assume that $P(g1/d_k) = P(g2/d_k) = \ldots\ldots\ldots = P(gm/d_k)$

Thus we can see that by having appropriate probability assignments we can get over the problem of uncertainty in rule definitions, because in real world cases, the predicates have not only to be qualitatively 'true', or 'False' but also quantitatively 'true' or 'false'.

Chapter 3

CONCLUSION

The way the expert system development has been attempted, there's many shortcomings remain in the system. One reason for this is not using the expert system shell or some kind of front-end system. Many enhancements to the existing system can be suggested which are as follows :

3.1 Facility for Explanation

Using front-end systems to Prolog such as APES, i.e. Augmented Prolog for Expert Systems we can give a facility to the user to get an explanation as to how the system reached a certain conclusion. For example, in the system being discussed in this report, we could have given a command using a front-end such as APES :-

```
FIND (DECISION; DIVPETN, LIVSEP, INCLIVTOGETH,
                    MUTDISSOL)
```

where the symbols after semicolon are a set of conditions. The Prolog would have answered (the decision) as: DIVORCE BY MUTUAL CONSENT.

The interesting thing is that then, we could have asked a question such as WHY? and $\overset{g}{\text{f}}$ot the following kind of explanation :

```
To Deduce
MUTUAL CONSENT DIVORCE as the decision
I used the rule
The decision is MUTUAL CONSENT DIVORCE if
(1 The petition is presented jointly    AND
(2 Both live separately for more than one yr AND
 3 They have not been to live together   AND
 4 They have mutually ageed for dissolution)
I can show that
1 is true
2 is true
3 is true
4 is true
```

This is an extremely useful facility because, even in the simplest of prototypes, it is easy to forget the details of the rules being used, and it is better if the system can remind.

The kind of knowledge which we have been most concerned about till now is called the basic 'Decision Knowledge' - that is, the rules that are used more or less explicitly by the expert in reaching a decision. If in a knowledge based we are concerned not only about what to do, but also about the reasons why something should or should not be done, then we need a deeper level of knowledge which can be called 'support knowledge'. In the system implemented here, the support knowledge would give the system the ability to answer the questions :

"Why should divorce not be the decision?"

In addition, the system can also be invested with the capability to answer questions from the user as to why it asked a particular question.

3.2 Change in the System Flow

An another level of knowledge is called the 'Meta-Knowledge'. This affects the control of the decision-making process rather than the decision itself - that is, meta-rules are the rules that affect the way the inference engine uses decision rules. Here is an example of Meta-rules which can be applied to the expert systems discussed in the report

"If there are rules relevant to Judicial separation
and if there are rules not relevant to judicial separation
and IF the current goal is judicial separation
THEN use first any rules relevant to judicial separation".

3.3 Capability of Modification of rules

This could be one of the most difficult to implement. At each level of rule i.e. in the present case, the hypotheses, the sections, the subsections, the conditions and the grounds, we can have a parameter attached (to all the instances of each level) which can take the following values:

1 - Add

2 - Delete

3 - Modify

At a prompt from the user to add/delete/modify for a particular ground (elementary condition), we can either delete that condition or (if the option is 1 or 3), we can further ask the user to specify the new conditions which can be placed in the knowledge base instead of the earlier one. Then depending on 'Yes' or 'No', from the user we could go a level higher and effect the same kind of changes there. This way we can change the grounds, conditions or even hypotheses. Such an enhancement would make the expert system worthy of a real life environment as new rules and regulations are formulated and old rules modified very frequently.

In addition, we would include learning facilities and a flexible user interface, possibly even in a natural language.

Many of the enhancements which have been discussed are still subjects for research in Artificial Intelligence. These techniques have been barely out of the lab to give evidence of fairly broad applicability in the business world. Still, each of the techniques and tools must be made specific to each expert domain, if the true essence of expertise in that particular domain is to be captured.

LIST OF REFERENCES

* (Barr and Feigenbaum 1981). "Handbook of Artificial Intelligence". William Kaufmann, Los Altos, Ca.

* (Duda and Hart 1973) Duda, R., and Hart, P. "Pattern classification" and scene Analysis".

* Charniak and Medermott, "Artifical Intelligence".

* (Hayes - Roth, waterman and Lenat, "Building Expert Systems" Addison Wesley, New York, 1983.

* Schildt, "Advanced Turbo PROLOG". Osborne Mcgraw, Hill, 1987.

* S.T. Desai, "Princciples of Hindu Law".

* Robert Keller, "Expert system technology, development and applications".

* Sholom M. Weiss & casimir A. Kullikowski, "A practical guide to designing expert systems".

* Carbonell, J.G., Michalski, R.S., & Mitchell, T.M, 1983, |"An overview of Machine learning.

* Cohen,P.R. & Feigenbum, E.A. 1982, "The Handbook of Artificial Intelligence."

A)   /*   APPENDIX   */

```
/* THIS SYSTEM FOLLOWS THE REASONING  */
/* OF A LAWYER IN ARRIVING AT A CONCLUSION*/
/* ON DIVORCE PETITIONS */
/*  PART A:DECLARATORY AND QUERY MANAGEMENT PART */

domains

decsn,grnds =symbol

query = string

answer =char


database

    xpositive(grnds)

    xnegative(grnds)

predicates

   hypothesis(decsn)

   sec(grnds)

   subsec(grnds)

   grnds(grnds)

   condtn(grnds)

   method(grnds)

   response(answer)

   go

   positive(query,grnds)

   clear_facts

   remember(grnds,answer)

   ask(query,grnds,answer)

clauses

   go:- clear_facts,

        clearwindow,
        hypothesis(Decsn),!,
        write ("THE DECISION IS--" ,Decsn     ), nl,
        clear_facts.
   positive ( _ , Grnds):-
             xpositive (Grnds),!.
   positive (Query , Grnds):-
              not  (xnegative(Grnds)),
              ask  (Query,Grnds,Answer),
              Answer = 'y'.
   ask(Query,Grnds,Answer):-
             write(Query),  nl,
             readchar(Answer),
             write(Answer),nl,
             remember(Grnds,Answer).
   remember(Grnds,'y'):-
             asserta(xpositive(Grnds)).
             remember(Grnds,'n'):-
             asserta(xnegative(Grnds)).
   clear_facts:-
             retract(xpositive(_)),fail.
   clear_facts:-
             retract(xnegative(_)),fail.
             clear_facts.



      /*  PART B: KNOWLEDGE-BASE PART */


  hypothesis(divor):-
            method(eithpetn),
            sec(one);
            method(eithpetn).
```

```
          sec(onea);
          method(wifepetn),
          sec(two).
hypothesis(judsep):-
          method(eithpetn),
          condtn(dashone).
hypothesis(mutcons):-
          subsec(one);
          subsec(two).

 subsec(one):-
          grnds(divpetn),
          grnds(livsep),
          grnds(inclivtogeth),
          grnds(mutdissol).

 subsec(two):-
          grnds(motnboth),
          grnds(notwith),
          grnds(satiscort).
 condtn(dashone):-
          grnds(adultery);
          grnds(cruelty);
          grnds(desrn);
          grnds(incunsmind);
          grnds(incvirlep);
          grnds(venrldis).

condtn(two):-
          grnds(morwif);
          grnds(giltrapsod);
          grnds(cohabnores);

        grnds(befage),
          grnds(reauo).
```

```
sec(one):-
          grnds(convrsn);
          grnds(relgordr);
          grnds(nohrdliv);
          condtn(dashone).
sec(onea):-
          grnds(noresumcohab),
          grnds(norestconj).
sec(two):-
          method(wifepetn),
          condtn(two).
method(eithpetn):-
     positive("has the petition been moved by either of the spouses(y/n)?"
                    ,eithpetn).
method(wifepetn):-
     positive("has the petition been moved by the wife for dissolution
     of marriage(y/n)?"   ,wifepetn).
grnds(cruelty):-
    grnds(iltretsuff);
    grnds(iltretapp).
grnds(desrn):-
     grnds(mindurn),
     grnds(resncos),
     grnds(consent).
grnds(convrsn):-
     grnds(abdrelg),
     grnds(adoprelg).
grnds(incvirlep):-
     grnds(virul),
     grnds(incur).
grnds(incursmind):-
     grnds(incurs);
```

```
      grnds(incomdev);
      grnds(psycho).
  grnds(venrldis):-
    grnds(venrl),
      grnds(commun).
  grnds(relgordr):-
      grnds(renwrld),
      grnds(perfrite).



     /*  PART C: USER-INTERFACE PART  */

 grnds(nohrdliv):-
    positive("has the other spouse not been heard of being alive
         for 7 years(y/n)?"
          ,nohrdliv).
 grnds(adultery):-
   positive("has the other spouse committed adultery(y/n)?",adultery).

  grnds(incuns):-
     positive("is the other spouse incurably of unsound mind(y/n)?",incun
  grnds(mentil):-
     positive("is the other spouse continuously afflicted with
          mental illness(y/n)",mentil).
  grnds(incomdev):-
     positive("does the other socuse have an  incomplete
          development of mind(y/n)?"
                   ,incomdev).
 grnds(psycho):-
   positive("is the other spouse afflicted with a psychological
      disorder of continuous nature(y/n)?"
              ,psycho).
   grnds(iltretsuff):-
     positive("has there been ill-treatment by other spouse causing
       suffering in body or mind(y/n)"
           ,iltretsuff).
   grnds(iltretapp):-
     positive("has there been ill-treatment by other spouse causing
   apprehension that further cohabitation will be harmful(y/n)"
        ,iltretapp).
   grnds(morwif):-
     positive("does the husband have more than one wife living(y/n)?"

        ,morwif).
   grnds(giltrapsod):-
     positive("is the husband guilty of rape or sodomy(y/n)?"
         ,giltrapsod).
   grnds(cohabnores):-
   positive("has the cohabitation not been resumed for ONE YEAR
  after passing decree awarding maintenance to wife(y/n)?"
       ,cohabnores).
   grnds(befage):-
    positive("was the girl's marriage solemnised before attaining
   the age of 15 YEARS (y/n)?"
       ,befage).
   grnds(repud):-
    positive("has the girl repudiated the marriage before attainin
   the age of 18 YEARS(y/n)?"
       ,repud).
  grnds(mindurn):-
   positive("has the desertion beenfor a continuous period of
  NOT LESS THAN 2 YEARS JUST BEFORE PETITION(y/n)?"
     ,mindurn).
 grnds(resncos):-
  positive("has there been desertion without reasonable cause(y/n)
     ,resncos).
```

```
grnds(consent):-
 positive("has there been desertion without consent or wish of
the petitioner(y/n)?"
  ,consent).
 grnds(abdrelg):-
  positive("has the respondent abdicated his/her religion by a
 clear act or renunciation(y/n)?"
   ,abdrelg).
grnds(adoprelg):-
 positive("has the respondent adopted the other religion by undergoing
formal conversion(y/n)?"
  ,adoprelg).
grnds(venrl):-
 positive("does the respondent have a veneral disease(y/n)?"
   ,venrl).
grnds(commun):-
 positive("is the disease of communicative form(y/n)?"
   ,commun).
 grnds(renwrld):-
  positive("has the respondent renounced all worldly affairs
 by adopting a religious order(y/n)?"
   ,renwrld).
 grnds(perfrite):-
  positive("has the respondent performed the requisite rites
 of the particular religious order(y/n)?"
  ,perfrite).
  grnds(divpetn):-
   positive("have both spouses jointly presented a petition
  for divorce(y/n)?"
    ,divpetn).
  grnds(livsep):-
   positive("have both spouses been living separately for
            [illegible] [illegible] presenting petition(y/n)?"
 more than 1 year before presentation [illegible]
   ,livsep).

grnds(inclivtogeth):-
  positive("they have not been able to live together(y/n)?"
    ,inclivtogeth).
grnds(mutdissol):-
  positive("they have mutually agreed for dissolution(y/n)?"
   ,mutdissol).
grnds(motnboth):-
  positive("a petition is made by both parties after a period
>6 months of date of petion and <18 months of date of petition(Y/n)"
   ,motnboth).
grnds(notwith):-
  positive("petition is not withdrawn inthe meantime(y/n)?"
    ,notwith).
grnds(satiscort):-
  positive("court feels that the averments in the petition are true(y/n)
   ,satiscort).
grnds(virul):-
  positive("is the other spouse suffering from virulent(i.e. malignant
or venomous)leprosy(y/n)?"
   ,virul).
grnds(incur):-
  positive("is it incurable(y/n)?"
   ,incur).


   response(Answer):-
            readchar(Answer),
            write(Answer),nl.


  /**************************************************************/
```