UNIT 1 EXPERT SYSTEMS

Structure **Page Nos.**

1.0 Introduction 5
1.1 Objectives 5
1.2 An Introduction to Expert Systems 6
1.3 Concept of Planning, Representing and using Domain Knowledge 7
1.4 Knowledge Representation Schemes 7
  1.4.1 Semantic Networks
  1.4.2 Frames
  1.4.3 Proposition and Predicate Logic
  1.4.4 Rule Based Systems
    1.4.4.1 Forward Chaining Systems
    1.4.4.2 Backward Chaining Systems
    1.4.4.3 Probability Certainty Factors in Rule Based Systems
1.5 Examples of Expert Systems: MYCIN, COMPASS 24
1.6 Expert System Building Tools 25
  1.6.1 Expert System Shells
    1.6.1.1 Knowledge Base
    1.6.1.2 Knowledge Acquisition Subsystem (Example: COMPASS)
    1.6.1.3 Inference Engine
    1.6.1.4 Explanation Sub-system (Example: MYCIN)
    1.6.1.5 User Interface
    1.6.1.6 An Expert System shell: EMYCIN
1.7 Some Application Areas of Expert Systems 32
1.8 Summary 32
1.9 Solutions/Answers 33
1.10 Further Readings 35

1.0 INTRODUCTION

Computer Science is the study of how to create models that can be represented in and executed by some computing equipment. A number of new types of problems are being solved almost everyday by developing relevant models of the domains of the problems under consideration. One of the major problems which humanity encounters is in respect of scarcity of human experts to handle problems from various domains of human experience relating to domains including domains those of health, education, economic welfare, natural resources and the environment. In this respect, the task for a computer scientist is to create, in addition to a model of the problem domain, a model of an expert of the domain as problem solver who is highly skilled in solving problems from the domain under consideration. The field of Expert Systems is concerned with creating such models. The task includes activities related to eliciting information from the experts in the domain in respect of how they solve the problems, and activities related to codifying that information generally in the form of rules. This unit discusses such issues about the design and development of an expert system.

1.1 OBJECTIVES

After going through this unit, you should be able to:

- discuss the various knowledge representation scheme;
- tell about some of the well-known expert systems;
- enumerate and use of various tools for building expert systems;
- explain the various expert system shells, and
- discuss various application areas for expert systems.

1.2 AN INTRODUCTION TO EXPERT SYSTEMS

First of all we must understand that an expert system is nothing but a computer program or a set of computer programs which contains the knowledge and some inference capability of an expert, most generally a human expert, in a particular domain. As expert system is supposed to contain the capability to lead to some conclusion based on the inputs provided, information it already contains and its processing capability, an expert system belongs to the branch of Computer Science called Artificial Intelligence.

Mere possessing an algorithm for solving a problem is not sufficient for a program to be termed an expert system, **it must also possess knowledge** i.e., if there is an expert system for a particular domain or area and if it is fed with a number of questions regarding that domain then sooner or later we can expect that these questions will be answered. So we can say that the knowledge contained by an expert system must contribute towards solving the problems for which it has been designed.

Also knowledge in a expert system must be regarding a **specific domain**. As a human being cannot be an expert in every area of life, similarly, an expert system which tries to simulate the capabilities of an expert also works in a particular domain. Otherwise it may be require to possess potentially infinite amount of knowledge and processing that knowledge in finite amount of time is an impossible task.

Taking into consideration all the points which have been discussed above, let us try to give one of the many possible definitions of an Expert System:

An Expert System is a computer program that possesses or represents knowledge in a particular domain, has the capability of processing/ manipulating or reasoning with this knowledge with a view to solving a problem, giving some achieving or to achieve some specific goal.

An expert system may or may not provide the complete expertise or functionality of a human expert but it must be able to assist a human expert in fast decision making. The program might interact with a human expert or with a customer directly.

Let us discuss some of the basic properties of an expert system:

- It tries to simulate human reasoning capability about a specific domain rather than the domain itself. This feature separates expert systems from some other familiar programs that use mathematical modeling or computer animation. In an expert system the focus is to emulate an expert’s knowledge and problem solving capabilities and if possible, at a faster rate than a human expert.
- It perform reasoning over the acquired knowledge, rather than merely performing some calculations or performing data retrieval.
- It can solve problems by using heuristic or approximate models which, unlike other algorithmic solutions are not guaranteed to succeed.

AI programs that achieve expert-level competence in solving problems in different domains are more called **knowledge based systems**. **A knowledge-based system** is any system which performs a job or task by applying rules of thumb to a symbolic representation of knowledge, instead of employing mostly algorithmic or statistical methods. Often the term *expert systems* is reserved for programs whose knowledge base contains the knowledge used by human experts, in contrast to knowledge gathered from textbooks or non-experts. **But more often than not, the two terms, expert systems and knowledge-based systems are taken us synonyms.** Together they represent the most widespread type of *AI* application. The area of human intellectual endeavour to be captured in an expert system is sometimes called the task domain. **Task** refers to some goal-oriented, problem-solving activity. **Domain** refers to the area within which the task is being performed. Some of the typical tasks are diagnosis, planning, scheduling, configuration and design. For example, a program capable of conversing about the weather would be a knowledge-based system, even if that program did not have any expertise in meteorology, but an expert system must be able to perform weather forecasting.

Building a expert system is known as **knowledge engineering** and its practitioners are called **knowledge engineers**. It is the job of the knowledge engineer to ensure to make sure that the computer has all the knowledge needed to solve a problem. **The knowledge engineer must choose** one or more forms in which to represent the required knowledge i.e., s/he must choose **one or more knowledge representation schemes** (A number of knowledge representing schemes like semantic nets, frames, predicate logic and rule based systems have been discussed above. We would be sticking to rule based representation scheme for our discussion on expert systems). S/he must also ensure that the computer can use the knowledge efficiently by selecting from a handful of reasoning methods.

1.3 CONCEPT OF PLANNING, REPRESENTING AND USING DOMAIN KNOWLEDGE

From our everyday experience, we know that in order to solve difficult problems, we need to do some sort of planning. Informally, we can say that **Planning** is the process that exploits the structure of the problem under consideration for designing a sequence of actions in order to solve the problem under consideration.

The knowledge of nature and structure of the problem domain is essential for planning a solution of the problem under consideration. For the purpose of planning, the problem environments are divided into two categories, viz., classical planning environments and non-classical planning environments. The **classical** planning environments/domains are fully observable, deterministic, finite, static and discrete. On the other hand, **non-classical** planning environments may be only partially observable and/or stochastic. In this unit, we discuss planning only for classical environments.

1.4 KNOWLEDGE REPRESENTATION SCHEMES

One of the underlying assumptions in Artificial Intelligence is that **intelligent behaviour can be achieved through the manipulation of symbol structures** (representing bits of knowledge). One of the main issues in *AI* is to find appropriate representation of problem elements and available actions as symbol structures so that the representation can be used to intelligently solve problems. In *AI,* **an important criteria about knowledge representation schemes or languages is that they should support inference**. For intelligent action, the inferencing capability is essential in view of the fact that we can’t represent explicitly everything that the system might ever need to know–**somethings have to be left implicit, to be inferred/deduced by the system** as and when needed in problem solving.

In general, a good knowledge representation scheme should have the following features:

- It should allow us to express the knowledge we wish to represent in the language. For example, the mathematical statement: *Every symmetric and transitive relation on a domain, need not be reflexive* is not expressible in First Order Logic.
- It should allow new knowledge to be inferred from a basic set of facts, as discussed above.
- It should have well-defined syntax and semantics.

Some popular knowledge representation schemes are:

- **Semantic networks,**
- **Frames,**
- **First order logic, and,**
- **Rule-based systems.**

As semantic networks, frames and predicate logic have been discussed in previous blocks so we will discuss these briefly here. We will discuss the rule-based systems in detail.

1.4.1 Semantic Networks

Semantic Network representations provide a **structured knowledge representation**. In such a network, parts of knowledge are clustered into semantic groups. In semantic networks, the concepts and entities/objects of the problem domain are represented by nodes and relationships between these entities are shown by arrows, generally, by directed arrows. In view of the fact that semantic network **representation is a pictorial depiction** of objects, their attributes and the relationships that exist between these objects and other entities. A semantic net is just a graph, where the nodes in the graph represent concepts, and the arcs are labeled and represent binary relationships between concepts. These networks provide a more natural way, as compared to other representation schemes, for mapping to and from a natural language.

For example, the fact (a piece of knowledge): ***Mohan struck Nita in the garden with a sharp knife last week***, is represented by the semantic network shown in *Figure 4.1*.

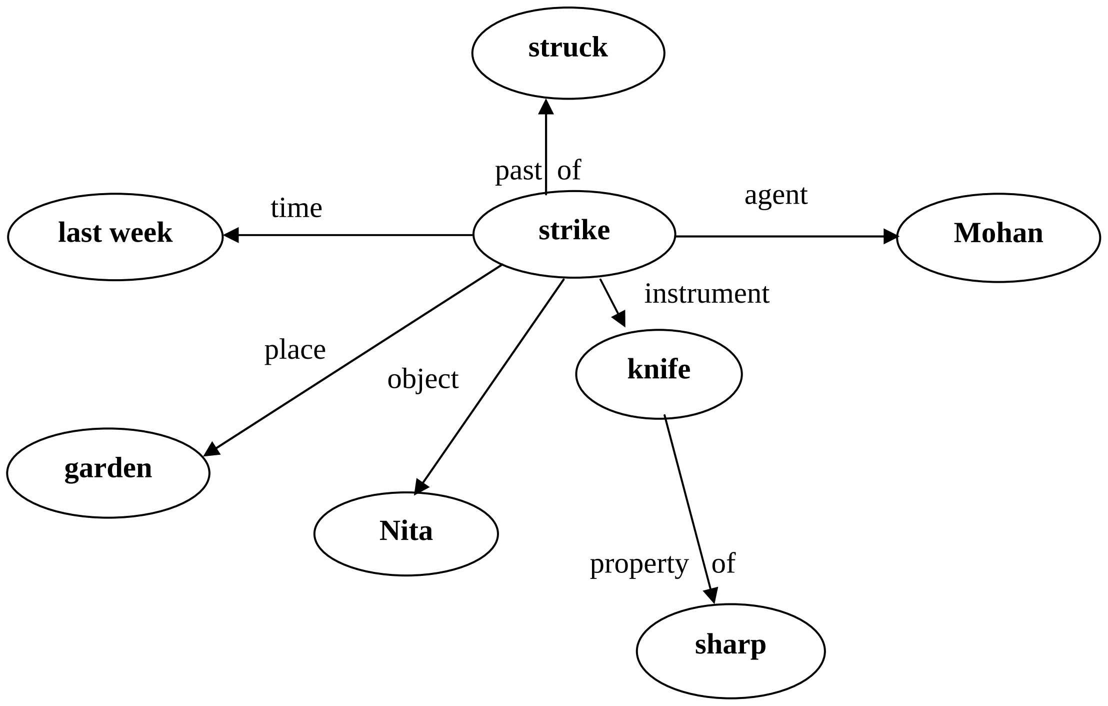


Figure 1.1 Semantic Network of the Fact in Bold

The two most important relations between concepts are *(i) subclass* relation between a class and its superclass, and *(ii) instance* relation between an object and its class. Other relations may be *has-part*, *color* etc. As mentioned earlier, relations are indicated by labeled arcs.

As information in semantic networks is clustered together through relational links, the knowledge required for the performance of some task is generally available within short spatial span of the semantic network. This type of knowledge organisation in some way, resembles the way knowledge is stored and retrieved by human beings.

Subclass and *instance* relations allow us to use **inheritance** to infer new facts/relations from the explicitly represented ones. We have already mentioned that **the graphical portrayal of knowledge in semantic networks,** being visual, is easier than other representation schemes for the human beings to comprehend. This fact helps the human beings to guide the expert system, whenever required. This is perhaps the reason for the popularity of semantic networks.

Exercise 1: Draw a semantic network for the following English statement:
Mohan struck Nita and Nita’s mother struck Mohan.

1.4.2 Frames

Frames are a variant of semantic networks that are one of the popular ways of representing non-procedural knowledge in an expert system. In a frame, all the information relevant to a particular concept is stored in a single complex entity, called a frame. Frames look like the data structure, record. Frames support inheritance. They are often used to capture knowledge about *typical* objects or events, such as a car, or even a mathematical object like rectangle. As mentioned earlier, a frame is a structured object and different names like *Schema, Script*, *Prototype*, and even *Object* are used in stead of frame, in computer science literature.

We may represent some knowledge about a lion in frames as follows:

```
Mammal :
  Subclass       :  Animal
  warm_blooded   :  yes

Lion :
  subclass       :  Mammal
  eating-habbit  :  carnivorous
  size           :  medium

Raja :
  instance       :  Lion
  colour         :  dull-Yellow
  owner          :  Amar Circus

Sheru :
  instance       :  Lion
  size           :  small
```

A particular frame (such as Lion) has a number of *attributes* or *slots* such as *eating-habit* and *size*. Each of these slots may be filled with particular values, such as the *eating-habit* for lion may be filled up as *carnivorous*.

Sometimes a slot contains additional information such as how to apply or use the slot values. Typically, a slot contains information such as *(attribute, value)* pairs, default values, conditions for filling a **slot**, pointers to other related frames, and also procedures that are activated when needed for different purposes.

In the case of frame representation of knowledge, **inheritance is simple** if an object has a single parent class, and if each slot takes a single value. For example, if a mammal is warm blooded then automatically a lion being a mammal will also be warm blooded.

But **in case of multiple inheritance** i.e., in case of an object having more than one parent class, we have to decide which parent to inherit from. For example, a lion may inherit from "wild animals" or "circus animals". In general, both the slots and slot values may themselves be frames and so on.

Frame systems are pretty complex and sophisticated knowledge representation tools. This representation has become so popular that special high level frame based representation languages have been developed. Most of these languages use LISP as the host language. It is also possible to represent frame-like structures using object oriented programming languages, extensions to the programming language LISP.

Exercise 2: Define a frame for the entity *date* which consists of *day*, *month* and *year*. each of which is a number with restrictions which are well-known. Also a procedure named *compute-day-of-week* is already defined.

1.4.3 Proposition and Predicate Logic

Symbolic logic may be thought of as a formal language, which incorporates a precise system for reasoning and deduction. Propositional logic and predicate logic are two well-known forms of symbolic logic. Propositional logic deals with propositions or statements.

A proposition or a statement is a sentence, which can be assigned a truth-value *true* or *false*. For example, the statement:

The sun rises in the west, has a truth value *false*.

On the other hand, none of the following sentences can be assigned a truth-value, and hence none of these, is a statement or proposition:

(i) Ram must exercise regularly. (Imperative sentence)
(ii) Who was the first Prime Minister of India? (Interrogative sentence)
(iii) Please, give me that book. (Imperative sentence)
(iv) Hurry! We have won the trophy. (Exclamatory sentence).

Generally, the following **steps are followed for solving problems using propositional logic,** where the problems are expressed in English and are such that these can be solved using propositional logic **(PL):**

a) First problem statements in English are translated to formulae of Propositional logic

b) Next, some of the rules of inference in PL including the ones mentioned below, are used to solve the problem, if, at all, the problem under consideration is solvable by PL.

Some of the rules of Inference of Propositional Logic:

(i) Modus Ponens

$$\frac{\begin{array}{c}P\\ P \to Q\end{array}}{Q}$$

(The above notation is interpreted as: if we are given the two formulae P and P→Q of the propositional logic, then conclude the formula Q. In other words, if both the formulae P and P→Q are assumed to be true, then by modus ponens, we can assume the statement Q also as true.)

(ii) Chain Rule

$$\frac{\begin{array}{c}P \to Q\\ Q \to R\end{array}}{P \to R}$$

(iii) Transposition

$$\frac{P \to Q}{\sim Q \to \sim P}$$

Example

Given the following three statements:
(i) Matter always existed
(ii) If there is God, then God created the universe
(iii) If God created the universe, then matter did not always exist.

To show the truth of the statement: *There is no God.*

Solution:

Let us denote the atomic statements in the argument given above as follows:
M: Matter always existed
TG: There is God
GU: God created the universe.
Then **the given** statements in English, become respectively the formulae of PL:
(i) M
(ii) TG→GU
(iii) GU→ ~ M
(iv) **To show** ~ TG
Applying transposition to (iii) we get
(v) M→ ~GU
using (i) and (v) and applying Modus Ponens, we get
(vi) ~GU
Again, applying transposition to (ii) we get
(vii) ~GU→ ~TG
Applying Modus Ponens to (vi) and (vii) we get
(viii) ~TG
The formula (viii) is the same as formula (iv) which was required to be proved.

Exercise 3: Using prepositional logic, show that, if the following statements are assumed to be true:
(i) There is a moral law.
(ii) If there is a moral law, then someone gave it.

(iii) If someone gave the moral law, then there is God.
then the following statement is also true:
(iv) There is God.

The trouble with propositional logic is that it is unable to describe properties of objects and also it lacks the structure to express relations that exist among two or more entities. Further, propositional logic does not allow us to make generalised statements about classes of similar objects. However, in many situations, the explicit knowledge of relations between objects and generalised statements about classes of similar objects, are essentially required to solve problems. Thus, propositional logic has serious limitations while reasoning for solving real-world problems. For example, let us look at the following statements:

(i) All children more than 12 years old must exercise regularly.
(ii) Ram is more than 12 years old.

Now 'these statements should be sufficient enough to allow us to conclude: *Ram must exercise regularly*. However, in propositional logic, each of the above statement is indecomposable and may be respectively denoted by P and Q. Further, whatever is said inside each statement is presumed to be not visible. Therefore, if we use the language of propositional logic, we are just given two symbols, viz., P and Q, representing respectively the two given statements. However, from just two propositional formulae P and Q, it is not possible to conclude the above mentioned statement viz., *Ram must exercise regularly*. To draw the desired conclusion with a valid inference rule, it would be necessary to use some other language, including some extension of propositional logic.

Predicate Logic, and more specifically, ***First Order Predicate Logic (FOPL)*** is an extension of propositional logic, which was developed to extend the expressiveness of propositional logic. In addition to just propositions of propositional logic, the predicate logic uses predicates, functions, and variables together with variable quantifiers *(Universal and Existential quantifiers)* to express knowledge.

We already have defined the structure of formulae of FOPL and also have explained the procedure for finding the meaning of formulae in FOPL. Though, we have already explained how to solve problems using FOPL, yet just for recalling the procedure for solving problems using FOPL, we will consider below one example.

In this context, we may recall the **inference rules of FOPL.** The inference rules of PL including *Modus Ponens, Chain Rule and Rule of Transposition* are valid in FOPL also after suitable modifications by which formulae of PL are replaced by formulae of FOPL.

In addition to these inference rules, the **following four inference rules of FOPL,** that will be called Q_1, Q_2, Q_3 and Q_4, have no corresponding rules in PL. *In the following F denotes a predicate and x a variable/parameter:*

$$Q_1: \quad \frac{\sim (\exists x) F(x)}{(\forall x) \sim F(x)} \quad \textit{and} \quad \frac{(\forall x) \sim F(x)}{\sim (\exists x) F(x)}$$

The first of the above rules under Q_1, says:
From *negation of there exists x F (x),* we can infer *for all x not of F (x)*

Q$_2$: $\dfrac{\sim (\forall x) F(x)}{(\exists x) \sim F(x)}$ *and* $\dfrac{(\exists x) \sim F(x)}{\sim (\forall x) F(x)}$

The first of the above rules under Q$_2$, says:
From *negation of for all x F (x),* we can infer *there exists x such that not of F (x)*

Q$_3$: $\dfrac{(\forall x) F(x)}{F(a)}$, where *a* is (any) arbitrary element of the domain of *F*
The rule Q$_3$ is called **universal instantiation**

$Q^{|}_{3}$: $\dfrac{F(a),\ for\ arbitrary\ a}{(\forall x) F(x)}$

The rule is called **universal generalisation**

Q$_4$: $\dfrac{(\exists x) F(x)}{F(a)}$, where *a* is a particular (not arbitrary) constant.

This rule is also called **existential instantiation:**

$Q^{|}_{4}$: $\dfrac{F(a)\ for\ some\ a}{(\exists x) F(x)}$
The rule is called **existential generalisation**

Steps for using Predicate Calculus as a Language for Representing Knowledge

Step 1: Conceptualization: First of all, all the relevant entities and the relations that exist between these entities are explicitly enumerated. Some of the implicit facts like 'a person dead once is dead forever' have to be explicated.

Step 2: Nomenclature and Translation: Giving appropriate names to the objects and relations. And then translating the given sentences given in English to formulae in FOPL.

Appropriate names are essential in order to guide a reasoning system based on FOPL. It is well-established that no reasoning system is complete. In other words, a reasoning system may need help in arriving at desired conclusion.

Step 3: Finding appropriate sequence of reasoning steps, involving selection of appropriate rule and appropriate FOPL formulae to which the selected rule is to be applied, to reach the conclusion.

While solving problems with FOPL, generally, the proof technique is proof by contradiction. Under this technique, the negation of what is to be proved is also taken as one of the assumptions. Then from the given assumptions alongwith the new assumption, we derive a contradiction, i.e., using inference rules, we derive a statement which is negation of either an assumption or is negation of some earlier derived formula.

Next, we give an example to illustrate how FOPL can be used to solve problems expressed in English and which are solvable by using FOPL. However, the proposed solution does not use the above-mentioned method of contradiction.

Example

We are given the statements:

(i) No feeling of pain is publically observable
(ii) All chemical processes are publically observable
We are to prove that
(iii) No feeling of pain is a chemical process

Solution:

For translating the given statements (i), (ii) & (iii), let us use the notation:

F(x): x is an instance of feeling of pain
O(x): x is an entity that is publically observable
C(x): x is a chemical process.

Then (i), (ii) and (iii) in FOPL can be equivalently expressed as
(i) $(\forall x)\ (\ F(x) \rightarrow \sim O(x)\ )$
(ii) $(\forall x)\ (\ C(x) \rightarrow O(x)\ )$

To prove
(iii) $(\forall x)\ (\ F(x) \rightarrow \sim C(x)\ )$

From (i) using generalized instantiation, we get
(iv) $F(a) \rightarrow \sim O(a)$ for any arbitrary a.

Similarly, from (ii), using generalized instantiation, we get
(v) $C(b) \rightarrow O(b)$ for any arbitrary b.

From (iv) using transposition rule, we get
(vi) $O(a) \rightarrow \sim F(a)$ for any arbitrary a

As b is arbitrary in (v), therefore we can rewrite (v) as
(vii) $C(a) \rightarrow O(a)$ for any arbitrary a
From (vii) and (vi) and using chain rule, we get
(viii) $C(a) \rightarrow \sim F(a)$ for any arbitrary a

But as a is arbitrary in (viii), by generalized quantification, we get
(ix) $(\forall x)\ (\ C(x) \rightarrow \sim F(x))$

But (ix) is the same as (iii), which was required to be proved.

Problems with FOPL as a system of knowledge representation and reasoning:
FOPL is not capable of easily representing some kinds of information including information pieces involving

(i) properties of relations. For example, the mathematical statement:
Any relation which is symmetric & transitive may not be reflexive
is not expressible in FOPL.
(ii) linguistic variables like hot, tall, sweat.
For example: *It is very cold today,*
can not be appropriately expressed in FOPL.
(iii) different belief systems.
For example, *s/he know that he thinks India will win the match, but I think India will lose,*
also, can not be appropriately expressed in FOPL.

Some shortcomings in predicate calculus

Now, after having studied predicate logic as a knowledge representation scheme, we ask ourselves the inevitable question, **whether it can be used to solve real world problems and to what extent**.

Before we try to answer the above question, let us review some of the properties of logic reasoning systems including predicate calculus. We must remember that three important properties of any logical reasoning system are ***soundness***, ***completeness*** and ***tractability***. To be confident that an inferred conclusion is true we require soundness. To be confident that inference will eventually produce any true conclusion, we require completeness. To be confident that inference is feasible, we require tractability.

Now, as we have discussed above also, in predicate calculus resolution refutation as an inference procedure is **sound** and **complete**. Because in case that the well formed formula which is to be proved is not logically followed or entailed by the set of well formed formulas which are to be used as premises, **the resolution refutation procedure might never terminate**. Thus, resolution refutation is not a full decision procedure. Also there is not other procedure that has this property or that is fully decidable. **So predicate calculus is *semi-decidable*** (semi-decidable means that if the set of input (well formed formulas) do not lead to the conclusion then the procedure will never stop or terminate).

But the situation is worse than this, as even on problems for which resolution refutation terminates, the procedure is **NP-hard** – as is any sound and complete inference procedure for the first-order predicate calculus i.e., it may take exponentially large time to reach a conclusion.

How to tackle these problems:

People who have done research in Artificial Intelligence have shown various ways: **First**, they say that we should not insist on the property of soundness of inference rules. Now what does it mean – basically it means that sometimes or occasionally our rules might prove an “untrue formula”.

Second, they say that we should not insist on the property of completeness i.e., to allow use of procedures that are not guaranteed to find proofs of true formulas.

Third, they also suggest that we could use a language that is less expressive that the predicate calculus. For example, a language in which everything is expressed using only Horn Clauses (Horn clauses are those which have at most one positive literal).

1.4.4 Rule Based Systems

Rather than representing knowledge in a declarative and somewhat static way (as a set of statements, each of which is true), rule-based systems represent knowledge in terms of a set of rules each of which specifies the conclusion that could be reached or derived under given conditions or in different situations. A rule-based system consists of

(i) Rule base, which is a set of IF-THEN *rules*,
(ii) A bunch of *facts*, and
(iii) Some interpreter of the facts and rules which is a mechanism which decides which rule to apply based on the set of available facts. The interpreter also initiates the action suggested by the rule selected for application.

A Rule-base may be of the form:

R_1: If A is an animal and A barks, than A is a dog
F1: Rocky is an animal
F2: Rocky Barks

The rule-interpreter, after scanning the above rule-base may conclude: Rocky is a dog. After this interpretation, the rule-base becomes

R_1: If A is an animal and A barks, then A is a dog
F1: Rocky is an animal
F2: Rocky Barks
F3: Rocky is a dog.

There are two broad kinds of rule-based systems:

***forward chaining* systems**,
and ***backward chaining* systems**.

In a **forward** chaining system we start with the initial facts, and keep using the rules to draw new intermediate conclusions (or take certain actions) given those facts. The process terminates when the final conclusion is established. In a **backward** chaining system, we start with some goal statements, which are intended to be established and keep looking for rules that would allow us to conclude, setting new sub-goals in the process of reaching the ultimate goal. In the next round, the subgoals become the new goals to be established. The process terminates when in this process all the subgoals are given fact. Forward chaining systems are primarily **data-driven**, while backward chaining systems are **goal-driven**. We will discuss each in detail.

Next, we discuss in detail some of the issues involved in a rule-based system.

Advantages of Rule-base

A basic principle of rule-based system is that each rule is an independent piece of knowledge. In an IF-THEN rule, the IF-part contains all the conditions for the application of the rule under consideration. THEN-part tells the action to be taken by the interpreter. The interpreter need not search any where else except within the rule itself for the conditions required for application of the rule.

Another important consequence of the above-mentioned characteristic of a rule-based system is that no rule can call upon any other and hence rules are ignorant and hence independent, of each other. This gives a highly modular structure to the rule-based systems. Because of the highly modular structure of the rule-base, the rule-based system addition, deletion and modification of a rule can be done without any danger side effects.

Disadvantages

The main problem with the rule-based systems is that when the rule-base grows and becomes very large, then checking (i) whether a new rule intended to be added is redundant, i.e., it is already covered by some of the earlier rules. Still worse, as the rule- base grows, checking the consistency of the rule-base also becomes quite difficult. By consistency, we mean there may be two rules having similar conditions, the actions by the two rules conflict with each other.

Let us first define working memory, before we study forward and backward chaining systems.

Working Memory: A working is a representation, in which

- lexically, there are application –specific symbols.
- structurally, assertions are list of application-specific symbols.
- semantically, assertions denote facts.
- assertions can be added or removed from working memory.

1.4.4.1 Forward Chaining Systems

In a forward chaining system the facts in the system are represented in a **working memory** which is continually updated, so on the basis of a rule which is currently being applied, the number of facts may either increase or decrease. Rules in the system represent possible actions to be taken when specified conditions hold on items in the working memory–they are sometimes called **condition-action or antecedent-consequent rules**. The conditions are usually **patterns** that must match items in the working memory, while the actions usually involve **adding *or* deleting** items from the working memory. So **we can say that in forward chaining proceeds forward, beginning with facts, chaining through rules, and finally establishing the goal**. Forward chaining systems usually represent rules in standard implicational form, with an antecedent or condition part consisting of positive literals, and a consequent or conclusion part consisting of a positive literal.

The interpreter controls the application of the rules, given the working memory, thus controlling the system's activity. It is based on a cycle of activity sometimes known as a **recognize-act** cycle. The system first checks to find all the rules whose condition parts are satisfied i.e., the those rules which are applicable, given the current state of working memory (**A rule is applicable if** each of the literals in its antecedent i.e., the condition part can be unified with a corresponding fact using consistent substitutions. This restricted form of unification is called pattern matching). It then selects one and performs the actions in the action part of the rule which may involve addition or deleting of facts. The actions will result in a new i.e., updated working memory, and the cycle starts again (**When more than one rule is applicable**, then some sort of external conflict resolution scheme is used to decide which rule will be applied. But when there are a large numbers of rules and facts then the number of unifications that must be tried becomes prohibitive or difficult). This cycle will be repeated until either there is no rule which fires, or the required goal is reached.

Rule-based systems vary greatly in their details and syntax, let us take the following example in which we use forward chaining :

Example

Let us assume that the working memory initially contains the following facts :

(day monday)

(at-home ram)

(does-not-like ram)

Let, the existing set of rules are:

R1 : IF (day monday)
THEN ADD to working memory the fact : (working-with ram)

R2 : IF (day monday)
THEN ADD to working memory the fact : (talking-to ram)

R3 : IF (talking-to X) AND (working-with X)
THEN ADD to working memory the fact : (busy-at-work X)

R4 : IF (busy-at-work X) OR (at-office X)
THEN ADD to working memory the fact : (not-at-home X)

R5 : IF (not-at-home X)
THEN DELETE from working memory the fact : (happy X)

R6 : IF (working-with X)
THEN DELETE from working memory the fact : (does-not-like X)

Now **to start the process of inference through forward chaining**, the rule based system will first search for all the rule/s whose antecedent part/s are satisfied by the current set of facts in the working memory. For example, in this example, we can see that the rules R1 and R2 are satisfied, so the system will chose one of them using its conflict resolution strategies. Let the rule R1 is chosen. So (working-with ram) is added to the working memory (after substituting "ram" in place of X). So working memory now looks like:

(working-with ram)

(day monday)

(at-home ram)

(does-not-like ram)

Now this cycle begins again, the system looks for rules that are satisfied, it finds rule R2 and R6. Let the system chooses rule R2. So now (taking-to ram) is added to working memory. So now working memory contains following:

(talking-to ram)

(working-with ram)

(day monday)

(at-home ram)

(does-not-like ram)

Now in the next cycle, rule R3 fires, so now (busy-at-work ram) is added to working memory, which now looks like:

(busy-at-work ram)

(talking-to ram)

(working-with ram)

(day monday)

(at-home ram)

(does-not-like ram)

Now antecedent parts of rules R4 and R6 are satisfied. Let rule R4 fires, so (not-at-home, ram) is added to working memory which now looks like :

(not-at-home ram)

(busy-at-work ram)

(talking-to ram)

(working-with ram)

(day monday)

(at-home ram)

(does-not-like ram)

In the next cycle, rule R5 fires so (at-home ram) is removed from the working memory :

(not-at-home ram)

(busy-at-work ram)

(talking-to ram)

(working-with ram)

(day monday)

(does-not-like ram)

The forward chining will continue like this. But we have to be sure of one thing, that the ordering of the rules firing is important. A change in the ordering sequence of rules firing may result in a different working memory.

Some of the conflict resolution strategies which are used to decide which rule to fire are given below:

- Don’t fire a rule twice on the same data.
- Fire rules on more recent working memory elements before older ones. This allows the system to follow through a single chain of reasoning, rather than keeping on drawing new conclusions from old data.
- Fire rules with more specific preconditions before ones with more general preconditions. This allows us to deal with non-standard cases.

These strategies may help in getting reasonable behavior from a forward chaining system, but **the most important thing is how should we write the rules**. They should be carefully constructed, with the preconditions specifying as precisely as possible when different rules should fire. Otherwise we will have little idea or control of what will happen.

1.4.4.2 Backward Chaining Systems

In forward chining systems we have seen how rule-based systems are used to draw new conclusions from existing data and then add these conclusions to a working memory. **The forward chaining approach is most useful when** we know all the initial facts, but we don’t have much idea what the conclusion might be.

If we know what the conclusion would be, or have some specific hypothesis to test, forward chaining systems may be inefficient. In forward chaining we keep on moving ahead until no more rules apply or we have added our hypothesis to the working memory. But in the process the system is likely to do a lot of additional and irrelevant work, adding uninteresting or irrelevant conclusions to working memory. Let us say that in the example discussed before, suppose we want to find out whether “ram is at home”. We could repeatedly fire rules, updating the working memory, checking each time whether **(at-home ram)** is found in the new working memory. But maybe we had a whole batch of rules for drawing conclusions about what happens when I'm working, or what happens on Monday–we really don't care about this, so would rather only have to draw the conclusions that are relevant to the goal.

This can be done by **backward chaining** from the goal state or on some hypothesized state that we are interested in. This is essentially how Prolog works. Given a goal state to try and prove, for example **(at-home ram),** the system will first check to see if the goal matches the initial facts given. If it does, then that goal succeeds. If it doesn't the system will look for rules whose conclusions i.e., actions match the goal. One such rule will be chosen, and the system will then try to prove any facts in the preconditions of the rule using the same procedure, setting these as new goals to prove. **We should note that a backward chaining system does not need to update a working memory.** Instead it needs to keep track of what goals it needs to prove its main hypothesis. So we can say that **in a backward chaining system, the reasoning proceeds "backward", beginning with the goal to be established, chaining through rules, and finally anchoring in facts**.

Although, in principle same set of rules can be used for both forward and backward chaining. However, **in backward chaining, in practice we may choose to write the rules slightly differently**. In backward chaining we are concerned with matching the conclusion of a rule against some goal that we are trying to prove. So the 'then or consequent' part of the rule is usually not expressed as an action to take (e.g., add/delete), but as a state which will be true if the premises are true.

To learn more, let us take a different example in which we use backward chaining (The system is used to identify an animal based on its properties stored in the working memory):

Example

1. Let us assume that the working memory initially contains the following facts:

(has-hair raja)	representing	the fact "raja has hair"
(big-mouth raja)	representing	the fact "raja has a big mouth"
(long-pointed-teeth raja)	representing	the fact "raja has long pointed teeth"
(claws raja)	representing	the fact "raja has claws"

Let, the existing set of rules are:

```
1.    IF (gives-milk X)
              THEN (mammal X)

2.    IF (has-hair X)
               THEN (mammal X)

3.    IF (mammal X) AND (eats-meat X)
              THEN (carnivorous X)

4.   IF (mammal X) AND (long-pointed-teeth X) AND (claws X)
               THEN (carnivorous X)

5.   IF (mammal X) AND (does-not-eat-meat X)
               THEN (herbivorous X)

6.    IF (carnivorous X) AND (dark-spots X)
```

THEN (cheetah, X)

7. IF (herbivorous X) AND (long-legs X) AND (long-neck X) AND (dark-spots X)
   THEN (giraffe, X)
8. IF (carnivorous X) AND (big-mouth X)
   THEN (lion, X)

9. IF (herbivorous X) AND (long-trunk X) AND (big-size X)
   THEN (elephant, X)

10. IF (herbivorous, X) AND (white-color X) AND ((black-strips X)
    THEN (zebra, X)

Now to start the process of inference through backward chaining, the rule based system will first form a hypothesis and then it will use the antecedent – consequent rules (previously called condition – action rules) to work backward toward hypothesis supporting assertions or facts.

Let us take the initial hypothesis that "raja is a lion" and then reason about whether this hypothesis is viable using backward chaining approach explained below :

- The system searches a rule, which has the initial hypothesis in the consequent part that someone i.e., raja is a lion, which it finds in rule 8.

- The system moves from consequent to antecedent part of rule 8 and it finds the first condition i.e., the first part of antecedent which says that "raja must be a carnivorous".

- Next the system searches for a rule whose consequent part declares that someone i.e., "raja is a carnivorous", two rules are found i.e., rule 3 and rule 4. We assume that the system tries rule 3 first.

- To satisfy the consequent part of rule 3 which now has become the system's new hypothesis, the system moves to the first part of antecedent which says that X i.e., raja has to be mammal.

- So a new sub-goal is created in which the system has to check that "raja is a mammal". It does so by hypothesizing it and tries to find a rule having a consequent that someone or X is a mammal. Again the system finds two rules, rule 1 and rule 2. Let us assume that the system tries rule 1 first.

- In rule 1, the system now moves to the first antecedent part which says that X i.e., raja must give milk for it to be a mammal. The system cannot tell this because this hypothesis is neither supported by any of the rules and also it is not found among the existing facts in the working memory. So the system abandons rule 1 and try to use rule 2 to establish that "raja is a mammal".

- In rule 2, it moves to the antecedent which says that X i.e., raja must have hair for it to be a mammal. The system already knows this as it is one of the facts in working memory. So the antecedent part of rule 2 is satisfied and so the consequent that "raja is a mammal" is established.

- Now the system backtracks to the rule 3 whose first antecedent part is satisfied. In second condition of antecedent if finds its new sub-goal and in turn forms a new hypothesis that X i.e., raja eats meat.

- The system tries to find a supporting rule or an assertion in the working memory which says that “raja eats meat” but it finds none. So the system abandons the rule 3 and try to use rule 4 to establish that “raja is carnivorous”.

- In rule 4, the first part of antecedent says that raja must be a mammal for it to be carnivorous. The system already knows that “raja is a mammal” because it was already established when trying to satisfy the antecedents in rule 3.

- The system now moves to second part of antecedent in rule 4 and finds a new sub-goal in which the system must check that X i.e., raja has long-pointed-teeth which now becomes the new hypothesis. This is already established as “ raja has long-pointed-teeth” is one of the assertions of the working memory.

- In third part of antecedent in rule 4 the system’s new hypothesis is that “raja has claws”. This also is already established because it is also one the assertions in the working memory.

- Now as all the parts of the antecedent in rule 4 are established so its consequent i.e., “raja is carnivorous” is established.

- The system now backtracks to rule 8 where in the second part of the antecedent says that X i.e., raja must have a big-mouth which now becomes the new hypothesis. This is already established because the system has an assertion that “raja has a big mouth”.

- Now as the whole antecedent of rule 8 is satisfied **so the system concludes that “raja is a lion”.**

We have seen that the system was able to work backward through the antecedent – consequent rules, using desired conclusions to decide that what assertions it should look for and ultimately establishing the initial hypothesis.

How to choose the type of chaining among forward or backward chaining for a given problem ?

Many of the rule based deduction systems can chain either forward or backward, but which of these approaches is better for a given problem is the point of discussion.

First, let us learn some basic things about rules i.e., **how a rule relates its input/s (i.e., facts) to output/s (i.e., conclusion)**. Whenever in a rule, a particular set of facts can lead to many conclusions, the rule is said to have a high degree of **fan out**, and a strong candidate of backward chaining for its processing. On the other hand, whenever the rules are such that a particular hypothesis can lead to many questions for the hypothesis to be established, the rule is said to have a high degree of fan in, and a high degree of **fan in** is a strong candidate of forward chaining.

To summarize, the following points should help in choosing the type of chaining for reasoning purpose :

- If the set of facts, either we already have or we may establish, can lead to a large number of conclusions or outputs , but the number of ways or input paths to reach that particular conclusion in which we are interested is small, then **the degree of fan out is more than degree of fan in. In such case, backward chaining is the preferred choice.**

- But, if the number of ways or input paths to reach the particular conclusion in which we are interested is large, but the number of conclusions that we can reach using the facts through that rule is small, then **the degree of fan in is more than the degree of fan out. In such case, forward chaining is the preferred choice**.

- For case where **the degree of fan out and fan in are approximately same**, then in case if not many facts are available and the problem is check if one of the many possible conclusions is true**, backward chaining is the preferred choice**.

1.4.4.3 Probability Certainty Factors in Rule Based System

Rule based systems usually work in domains where conclusions are rarely certain, even when we are careful enough to try and include everything we can think of in the antecedent or condition parts of rules.

Sources of Uncertainty

Two important sources of uncertainty in rule based systems are:

✓ The theory of the domain may be vague or incomplete so the methods to generate exact or accurate knowledge are not known.
✓ Case data may be imprecise or unreliable and evidence may be missing or in conflict.

So even though methods to generate exact knowledge are known but they are impractical due to lack or data, imprecision or data or problems related to data collection.

So ***rule based deduction system developers often build some sort of certainty or probability computing procedure on and above the normal condition-action format of rules.*** Certainty computing procedures attach a **probability between 0 and 1** with each assertion or fact. Each probability reflects how certain an assertion is, whereas certainty factor of 0 indicates that the assertion is definitely false and certainty factor of 1 indicates that the assertion is definitely true.

Example 1: In the example discussed above the assertion (ram at-home) may have a certainty factor, say 0.7 attached to it.

Example 2: In MYCIN a rule based expert system (which we will discuss later), a rule in which statements which link evidence to hypotheses are expressed as decision criteria, may look like :

IF patient has symptoms s1,s2,s3 and s4
AND certain background conditions t1,t2 and t3 hold
THEN the patient has disease d6 with certainty 0.75

For detailed discussion on certainty factors, the reader may refer to probability theory, fuzzy sets, possibility theory, Dempster-Shafter Theory etc.

Exercise 4

In the “Animal Identifier System” discussed above use forward chaining to try to identify the animal called “raja”.

1.5 EXAMPLES OF EXPERT SYSTEMS: MYCIN, COMPASS

The first expert system we choose as and example is MYCIN, which is of the earliest developed expert systems. As another example of and expert system we briefly discuss COMPASS.

MYCIN (An expert system)

Like every one else, we are also tempted to discuss MYCIN, one of the earliest designed expert systems in Stanford University in 1970s.

MYCIN’s job was to diagnose and recommend treatment for certain blood infections. To do the proper diagnosis, it is required to grow cultures of the infecting organism which is a very time consuming process and sometime patient is in a critical state. So, doctors have to come up with quick guesses about likely problems from the available data, and use these guesses to provide a treatment where drugs are given which should deal with any type of problem.

So MYCIN was developed in order to explore how human experts make these rough (but important) guesses based on partial information. Sometimes the problem takes another shape, that an expert doctor may not available every-time every-where, in that situation also and expert system like MYCIN would be handy.

MYCIN represented its knowledge as a set of IF-THEN rules with certainty factors. One of the MYCIN rule could be like :

IF infection is primary-bacteremia AND the site of the culture is one of the sterile sites
AND the suspected portal of entry is the gastrointestinal tract
THEN there is suggestive evidence (0.8) that bacteroid infection occurred.

The 0.8 is the certainty that the conclusion will be true given the evidence. If the evidence is uncertain the certainties of the pieces of evidence will be combined with the certainty of the rule to give the certainty of the conclusion.

MYCIN has been written in Lisp, and its rules are formally represented as lisp expressions. The action part of the rule could just be a conclusion about the problem being solved, or it could be another lisp expression.

MYCIN is mainly **a goal-directed system**, using the **backward chaining** reasoning approach. However, to increase it reasoning power and efficiency MYCIN also uses various heuristics to control the search for a solution.

One of the strategy used by MYCIN is **to first ask the user a number of predefined questions that are most common** and which allow the system to rule out totally unlikely diagnoses. Once these questions have been asked, the system can then focus on particular and more specific possible blood disorders. It then uses backward chaining approach to try and prove each one. This strategy avoids a lot of unnecessary search, and is similar to the way a doctor tries to diagnose a patient.

The other strategies are related to the sequence in which rules are invoked. One of the strategy is simple i.e., given a possible rule to use, MYCIN first checks all the antecedents of the rule to see if any are known to be false. If yes, then there is no point using the rule. The other strategies are mainly related to the **certainty factors**. MYCIN first find the rules that have greater degree of certainty of conclusions, and abandons its search once the certainties involved get below a minimum threshold, say, 0.2.

There are three main stages to the interaction with MYCIN. In the first stage, initial data about the case is gathered so the system can come up with a broad diagnosis. In the second more directed questions are asked to test specific hypotheses. At the end of this section it proposes a diagnosis. In the third stage it asks questions to determine an appropriate treatment, on the basis of the diagnosis and facts related to the patient. After that it recommends some treatment. At any stage the user can ask why a question was asked or how a conclusion was reached, and if a particular treatment is recommended the user can ask if alternative treatments are possible.

MYCIN has been popular in expert system's research, but it also had a number of problems or shortcomings because of which a number of its derivatives like NEOMYCIN developed.

COMPASS (An expert system)

Before we discuss this let us understand the functionality of telephone company's switch. A telephone company's switch is a complex device whose circuitry may encompass a large part of a building. The goals of switch maintenance are to minimize the number of calls to be rerouted due to bad connections and to ensure that faults are repaired quickly. Bad connections caused due to failure of connection between two telephone lines.

COMPASS is an expert system which checks error messages derived from the switch's self test routines, look for open circuits, reduces the time of operation of components etc. To find the cause of a switch problem, it looks at a series of such messages and then uses it expertise. The system can suggest the running of additional tests, or the replacement of a particular component, for example, a relay or printed circuit board.

As expertise in this area was scarce, so it was a fit case for taking help an expert system like COMPASS (We will discuss later, how knowledge acquisition is done in COMPASS).

1.6 EXPERT SYSTEM BUILDING TOOLS

Expert system tools are designed to provide an environment for development of expert systems mainly through the approach of prototyping.

The expert systems development process is normally a mixture of prototyping and other incremental development models of software engineering rather than the conventional "waterfall model". Although using incremental development has a problem of integrating new functionality with the earlier version of expert system but the expert system development environments try to solve this problem by using modular representations of knowledge (some of the representation schemes like frames, rule based systems etc. have already been discussed above).

Software tools for development of expert systems mainly fall into the following Categories:

- **<u>Expert System Shells</u>:** These are basically a set of program and to be more specific - abstractions over one or more applications programs. One of the major examples is EMYCIN which is the rule interpreter of the famous expert system called MYCIN (a medical diagnostic system). EMYCIN also constitutes related data structures like knowledge tables and indexing mechanism over these tables. Some recent versions of EMYCIN like M.4 is a very sophisticated shell which combine the backward chaining of EMYCIN with frame – like data structures.

- **<u>High Level Programming Languages</u> :** These languages are basically used to provide a higher level of abstraction by hiding the implementation details because of which the programmer need not worry about efficiency of storing, accessing and manipulating data. One of the good examples is OPS5 rule language. But many of such languages are research tools and may not available commercially.

- **<u>Multiple programming environments</u> :** These provide a set of software modules to allow the user to use and mix a number of different styles of artificial intelligence programming. One of the examples is called LOOPS which combines rule based and object – oriented approaches of knowledge representation.

We will now keep our focus on “Expert System Shells”.

1.6.1 Expert System Shells

An expert system tool, or shell, is a software development environment containing the basic components of expert systems. Associated with a shell is a prescribed method for building applications by configuring and instantiating these components.

Expert system shells are basically used for the purpose of allowing non-programmers to take advantage of the already developed templates or shells and which have evolved because of the efforts of some pioneers in programming who have solved similar problems before. The core components of an expert systems are the knowledge base and the reasoning engine.

A generic expert system shell is shown below :

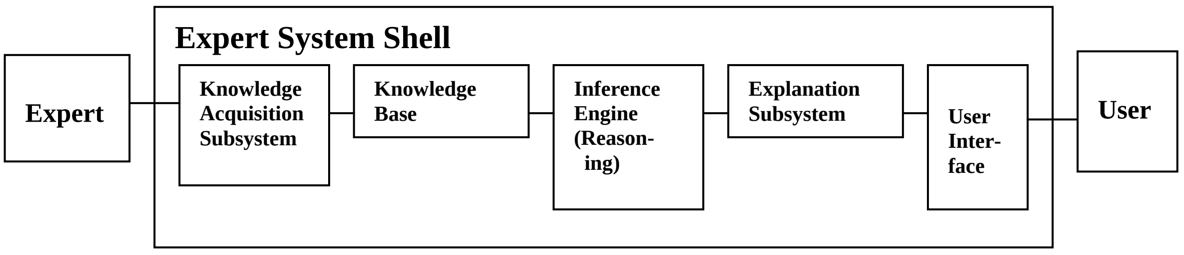


Figure 1.2: Components of Expert System Tool (Shell)

As we can see in the figure above, **the shell includes** the *inference engine, a knowledge acquisition subsystem, an explanation subsystem and a user interface.* When faced with a new problem in any given domain, we can find a shell which can provide the right support for that problem, so all we need is the knowledge of an expert. There are many commercial expert system shells available now, each one adequate for a different range of problems. Taking help of expert system shells to develop expert systems greatly reduces the cost and the time of development as compared to developing the expert system from the scratch.

Let us now discuss the components of a generic expert system shell. We will discuss about:

- **Knowledge Base**
- **Knowledge Acquisition Subsystem**
- **Inference Engine**
- **Explanation Subsystem**
- **User Interface**

1.6.1.1 Knowledge Base

It contains facts and heuristic knowledge. Developers try to use a uniform representation of knowledge as for as possible. There are many knowledge representation schemes for expressing knowledge about the application domain and some advance expert system shells use both frames (objects) and IF-THEN rules. In PROLOG the knowledge is represented as logical statements.

1.6.1.2 Knowledge Acquisition Subsystem

The process of capturing and transformation of potentially useful information for a given problem from any knowledge source (which may be a human expert) to a program in the format required by that program is the job of a knowledge acquisition subsystem. So we can say that these subsystem to help experts build knowledge bases.

As an expert may not be a computer literate, so capturing information includes interviewing, preparing questionnaires etc. which is a very slow and time consuming process. So **collecting knowledge needed to solve problems** and build the knowledge base has always been **the biggest bottleneck** in developing expert systems.

Some of the reasons behind the difficulty in collecting information are given below :

- *The facts and rules or principles of different domains cannot easily be converted into a mathematical or a deterministic model, the properties of which are known*. For example a teacher knows how to motivate the students but putting down on the paper, the exact reasons, causes and other factor affecting students is not easy as they vary with individual students.

- *Different domains have their own terminology* and it is very difficult for experts to communicate exactly their knowledge in a normal language.

- *Capturing the facts and principles alone is not sufficient to solve problems*. For example, experts in particular domains which information is important for specific judgments, which information sources are reliable and how problems can be simplified, which is based on personal experience. Capturing such knowledge is very difficult.

- *Commonsense knowledge found in humans continues to be very difficult to capture*, although systems are being made to capture it.

The idea of automated knowledge capturing has also been gaining momentum. Infact **"machine learning"** is one of the important research area for sometime now. The goal is that, a computing system or machine could be enabled to learn in order to solve problems like the way human do it.

Example (Knowledge acquisition in COMPASS).

As we already know that COMPASS is an expert system which was build for proper maintenance and troubleshooting of telephone company's switches.

Now, for knowledge acquisition, knowledge from a human expert is elicited. An expert explains the problem solving technique and a knowledge engineers then converts it into and if-then-rule. The human expert then checks if the rule has the correct logic and if a change is needed then the knowledge engineer reformulates the rule.

Sometimes, it is easier to troubleshoot the rules with pencil and paper (i.e., hand simulation), at least the first time than directly implementing the rule and changing them again and again.

Let us summarize the knowledge acquisition cycle of COMPASS :

- Extract knowledge from a human expert.
- Document the extracted knowledge.
- Test the new knowledge using following steps:

    - Let the expert analyze the new data.
    - Analyze the same data using pencil and paper using the documented knowledge.
    - Compare the results of the expert's opinion with the conclusion of the analysis using pencil and paper.
    - If there is a difference in the results, then find the rules which are the cause of discrepancy and then go back to rule 1 to gather more knowledge form the expert to solve the problem.
      Otherwise, exit the loop.

1.6.1.3 Inference Engine

An inference engine is used to perform reasoning with both the expert knowledge which is extracted from an expert and most commonly a human expert) and data which is specific to the problem being solved. Expert knowledge is mostly in the form of a set of IF-THEN rules. The case specific data includes the data provided by the user and also partial conclusions (along with their certainty factors) based on this data. In a normal forward chaining rule-based system, the case specific data is the elements in the working memory.

Developing expert systems involve knowing how knowledge is accessed and used during the search for a solution. Knowledge about what is known and, when and how to use it is commonly called **meta-knowledge**. In solving problems, a certain level of planning, scheduling and controlling is required regarding what questions to be asked and when, what is to be checked and so on.

Different strategies for using domain-specific knowledge have great effects on the performance characteristics of programs, and also on the way in which a program finds or searches a solution among possible alternatives. Most knowledge representations schemes are used under a variety of reasoning methods and research is going on in this area.

1.6.1.4 Explanation Subsystem (Example MYCIN)

An explanation subsystem allows the program to explain its reasoning to the user. The explanation can range from how the final or intermediate solutions were arrived at to justifying the need for additional data.

Explanation subsystems are important from the following points of view :

(i) **Proper use of knowledge:** There must be some for the satisfaction of knowledge engineers that the knowledge is applied properly even at the time of development of a prototype.
(ii) **Correctness of conclusions:** User's need to satisfy themselves that the conclusions produced by the system are correct.
(iii) **Execution trace:** In order to judge that the knowledge elicitation is proceeding smoothly and successfully, a complete trace of program execution is required.
(iv) **Knowledge of program behavior:** For proper maintenance and debugging, the knowledge of program behavior is necessary for the programmers.
(v) **Suitability of reasoning approach:** Explanation subsystems are necessary to ensure that reasoning technique applied is suitable to the particular domain.

Explanation in expert systems deals with the issue of control because the reasoning steps used by the programs will depend on how it searches for a solution.

Explanation subsystems are also related to evaluation as by checking the outputs produced by a system and after examining the trace of its actions performed while reasoning, it can be decided that whether or not a system is producing the right answer for right reasons. Without a good explanation subsystem, an expert will be unable to judge the system's performance or will be unable to find ways to improve it.

To give an idea of how an explanation subsystem works, let us have a simple example:

Example:

Suppose there is a simple rule based system **to diagnose the problems in a car**.

Let the system has following rules :

R1: IF engine-gets-petrol AND engine-turns-over
THEN spark-plug-fault

R2: IF NOT engine-turns-over AND NOT lights-come-on
THEN battery-problem

R3: IF NOT engine-turns-over AND lights-come-on
THEN starter-not-working

R4: IF fuel-tank-has-petrol
THEN engine-gets-petrol

We also assume that the system has no initial facts.

Explanation subsystems allow the user to ask ***why*** *it asked a particular question*, and ***how*** *it reached some conclusion*. These questions are answered by referring to the system goals, the rules being used, and any existing problem data.

Now let us focus on the example given above to see the explanation facilities provided, which involve a dialogue involving *why* and *how* questions and their explanations.

System : Does the fuel tank has gas?
User : Yes.

System : Does the engine turns over?
User : No

System : Do lights come on?
User : **Why?**

System : **I am trying to prove that there is a battery problem**. It has been established
that the engine does not turns over. Therefore if it is not the case that the lights
come on, then there is a battery problem.
Is it true that the lights come on?
User : No.

System : I conclude that there is a battery problem.
User : **How?**

System : This follows from Rule 2:
IF NOT engine-turns-over AND NOT lights-come-on was given by the user.

Providing such an explanation facility involves stating what rules are used in reaching conclusions, and using these records to compose explanations like the ones above. Giving simple explanations like those above is not very difficult, and is quite useful.

Explanation facilities in expert systems are sometimes not used, and sometimes they are not easily accepted by their users. There are a whole lot of reasons for this. One reason is that the explanations just reference the "surface" knowledge encoded in the rules, rather than providing the "deep" knowledge about the domain which originally motivated the rules but which is usually not represented. So, the system will say that it concluded something because of rule 5, but not explain what rule 5 intends to say. In the example given below, maybe the user needs to understand that both the lights and the starter use the battery, which is the underlying purpose of the second rule in this example. Another reason for the frequent failure of explanation facilities is the fact that, if the user fails to understand or accept the explanation, the system can't re-explain in another way (as people can). Explanation generation is a fairly large area of research, concerned with effective communication i.e., how to present things so that people are really satisfied with the explanation, and what implications does this have for how we represent the underlying knowledge.

Explanation Subsystem in MYCIN (An overview)

MYCIN is one of the first popular expert systems made for the purpose of medical diagnosis. Let us have a look at how the explanation subsystem in MYCIN works :
To explain the reasoning for deciding on a particular medical parameter's or symptom's value, it retrieves a set of rules and their conclusions. It allows the user to ask questions or queries during a consultation.

To answer the questions the system relies on its ability to display a rule invoked at any point during the consultation and also recording the order of rule invocations and associating them with particular events (like particular questions).

As the system using backward chaining so most of the questions belong to "Why" or "How" category. To answer "Why" questions, the system looks up the hierarchy (i.e., tree) of rules to see which goals the system is trying to achieve and to answer "Why" questions, the system must look down the hierarchy (i.e., tree) to find out that which sub-goals were satisfied to achieve the goal.

We can see that explanation process is nothing but a search problem requiring tree traversal.

As MYCIN keeps track of the goal to sub-goal sequence of the computation, so it can answer questions like:

“Why did you ask that if the stain of the organism is gram negative ?”

In response to this, the system would quote the rules which states “gram negative staining” may be in conjunction with other conditions and would state that what it was trying to achieve.

Similarly, if there is “How” question like:

“How do say that Organism-2 might be proteus ?”

In its reply, MYCIN would state the rules that were applied in reaching this conclusions and their degree of certainty and what was the last question asked etc.

Thus we can see that because of the backward chaining approach, the system is able to answer “Why” and “How” questions satisfactorily. But the rule application would not be easy if the chains of reasoning are long.

1.6.1.5 User interface

It is used to communicate with the user. The user interface is generally not a part of the expert system technology, and was not given much attention in the past. However, it is now widely accepted that the user interface can make a critical difference in the utility of a system regardless of the system’s performance.

Now as an example, let us discuss and expert system shell called EMYCIN.

1.6.1.6 EMYCIN (An expert system shell)

EMYCIN provides a domain-independent framework or template for constructing and running any consultation programs. EMYCIN stands for “Empty MYCIN” or “Essential MYCIN” because it basically constitutes a MYCIN system minus its domain-specific medical knowledge. However, EMYCIN is something more than this, as it offers a number of software tools for helping expert system designers in building and debugging performance programs.

Some characteristics of EMYCIN are:

- It constitutes an abbreviated rule language, which uses ALGOL-like notation and which is easier than LISP and is more concise than the English subset used by MYCIN.
- It uses backward chaining which is similar to MYCIN.
- It indexes rules, in-turn organising them into groups, based on the parameters which are being referenced.
- It has an interface for system designer which provides tools for displaying, editing and partitioning rules, editing knowledge held in tables, and also running rule sets on sets of problems. As part of system designer’s interface, EMYCIN

also included a knowledge editor (a program) called TEIRESIAS whose job was to provide help for the development and maintenance of large knowledge bases.

- It also has a user interface which allows the user to communicate with the system smoothly.

1.7 SOME APPLICATION AREAS OF EXPERT SYSTEMS

The scope of applications of expert systems technology to practical problems is so wide that it is very difficult to characterize them. The applications find their way into most of the areas of knowledge work. Some of the main categories of applications or an expert system are given below.

- **Diagnosis and Troubleshooting:** This class comprises systems that deduce faults and suggest corrective actions for a malfunctioning device or process. Medical diagnosis was one of the first knowledge areas to which ES technology was applied, but use of expert systems for solving and diagnosis of engineered systems has become common.

- **Planning and Scheduling:** Systems that fall into this class analyze a set of one or more potentially complex and interacting goals in order to determine a set of actions to achieve those goal. This class of expert systems has great commercial potential. Examples include scheduling of flights, personnel, manufacturing process planning etc.

- **Process Monitoring and Control:** Systems falling in this class analyze real-time data from physical devices with the goal of noticing errors, predicting trends, and controlling for both optimality and failure correction. Examples of real-time systems that actively monitor processes are found in the steel making and oil refining industries.

- **Financial Decision Making:** The financial services industry has also been using expert system techniques. Expert systems belonging to this category act as advisors, risk analyzers etc.

- **Knowledge Publishing:** This is a relatively new, but also potentially explosive area. The primary function of the expert system is to deliver knowledge that is relevant to the user's problem, in the context of the user's problem.

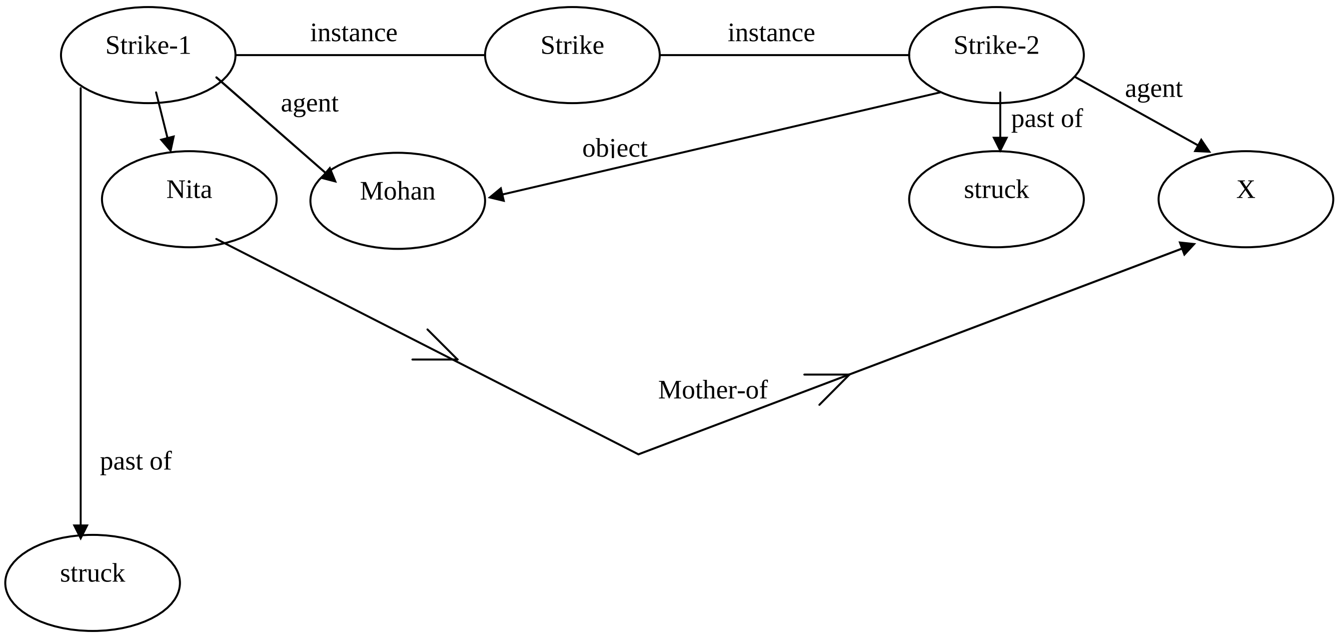
- **Design and Manufacturing:** These systems assist in the design of physical devices and processes, ranging from high-level conceptual design of abstract entities all the way to factory floor configuration of manufacturing processes.

1.8 SUMMARY

In this unit, we have discussed various issues in respect of expert systems. To begin with, in section 1.2 we define the concept of 'expert system'. In view of the fact that an expert system contains knowledge of a specific domain, in section 1.4, we discuss various schemes for representing the knowledge of a domain. Section 1.5 contains examples of some well-known expert systems. Tools for building expert systems are discussed in section 1.6. Finally, applications of expert systems are explained in section 1.7.

1.9 SOLUTIONS/ANSWERS

Ex. 1) In this case, it is not the same 'strike' action but two strike actions which are involved in the sentence. Therefore, we use 'strike' to denote a generic action of striking whereas 'strike-1' and 'strike-2' are its instances or members. Thus, we get the semantic network.



Ex. 2) (date
  (day (integer (1….31)))
  (month (integer (1…..12)))
  (year (integer (1…..10000)))
  (day-of-the-week (set (Mon Tue Wed Thu Fri Sat Sun)))
  (procedure (compute day-of-the-week (day month year))),

Where the procedure day-of-the-week takes three arguments.

Some problems in Semantic Networks and Frames:

There are problems in expressing certain kinds of knowledge when either semantic networks or frames are used for knowledge representation. For example, **it is difficult** although not impossible **to express disjunctions and hence implications, negations, and general non-taxonomic knowledge** (i.e., non-hierarchical knowledge) in these representations.

Ex. 3) In order to translate in PL, let us use the symbols:
ML: There is a Moral Law
SG: Someone Gave it (the word 'it' stands for moral law)
TG: There is God.
Using these symbols, the ***statements*** (i) to (iv) become the ***formulae*** (i) to (iv) of PL as given below:
(i) ML
(ii) ML→SG
(iii) SG→TG and

(iv) TG

Applying Modus Ponens to formulae (i) and (ii) we get the formula
(v) SG
Applying Modus Ponens to (v) and (iii), we get
(vi) TG
But formula (vi) is the same as (iv), which is required to be established. Hence the proof.

Ex. 4) Initially, the working memory contains the following assertions :

(has-hair raja)	representing	the fact "raja has hair"
(big-mouth raja)	representing	the fact "raja has a big mouth"
(long-pointed-teeth raja)	representing	the fact "raja has long pointed teeth"
(claws raja)	representing	the fact "raja has claws"

We start with one of the assertions about "raja" from the working memory.

Let us choose the first assertion : (raja has-hair)
Now we take this assertion and try to match the antecedent part of a rule. In the rule 2, the antecedent part is satisfied by substituting "raja" in place of X. So the consequent (mammal X) is established by replacing "raja" in place of X.

The working memory is now updated by adding the assertion (mammal raja).
So the working memory now looks like:

(mammal raja)

(has-hair raja)

(big-mouth raja)

(long-pointed-teeth raja)

(claws raja)

Now we try to match assertion (mammal raja) to the antecedent part of a rule. The first rule whose antecedent part supports the assertion is rule 3. So the control moves to the second part of rule 3, which says that (eats-meat X), but this is not found in any of the assertions present in working memory, so rule 3 fails.

Now, the system tries to find another rule which matches assertion (mammal raja), it find rule 4 whose first part of antecedent supports this. So the control moves to the second part of the antecedent in rule 4, which says that something i.e., X must have pointed teeth. This fact is present in the working memory in the form of assertion (long-pointed-teeth raja) so the control now moves to the third antecedent part of rule 4 i.e., something i.e., X must have claws. We can see that this is supported by the assertion (claws raja) in the working memory. Now, as the whole antecedent of rule 4 is satisfied so the consequent of rule 4 is established and the working memory is updated by the addition of the assertion (carnivorous raja), after substituting "raja" in place of X.

So the working now looks like:

(carnivorous raja)

(mammal raja)

(has-hair raja)

(big-mouth raja)

(long-pointed-teeth raja)

(claws raja)

Now in the next step, the system tries to match the assertion (carnivorous raja) with one of the rules in working memory. The first rule whose antecedent part matches this assertion is rule 6. Now as the first part of the antecedent in rule 6 matches with the assertion, the control moves to the second part of the antecedent i.e., X has dark spots. There is no assertion in working memory which supports this, so the rule 6 is aborted.

The system now tries to match with the next rule which matches the assertion (carnivorous raja). It finds rule 8 whose first part of antecedent matches with the assertion. So the control moves to the second part of the antecedent of rule 8 which says that something i.e., X must have big mouth. Now this is already present in the working memory in the form of assertion (big-mouth raja) so the second part and ultimately the whole antecedent of rule 8 is satisfied.

And, so the consequent part of rule 8 is established and the working memory is updated by the addition of the assertion (lion raja), after substituting “raja” in place of X.

The working memory now looks like:

(lion raja)

(carnivorous raja)

(mammal raja)

(has-hair raja)

(big-mouth raja)

(long-pointed-teeth raja)

(claws raja)

Hence, as the goal to be achieved i.e., “raja is a lion” is now part of the working memory in the form of assertion (lion raja), **so the goal is established** and processing stops.

1.10 FURTHER READINGS

1. Russell S. & Norvig P., *Artificial Intelligence A Modern Approach* (Second Edition) (Pearson Education, 2003).

2. Patterson D. W: *Introduction to Artificial Intelligence and Expert Systems* (Prentice Hall of India, 2001).