UNIT 6 RULE BASED SYSTEMS AND OTHER FORMALISM

Structure **Page Nos.**

6.0 INTRODUCTION

Computer Science is the study of how to create models that can be represented in and executed by some computing equipment. 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, and the concerned field relates to the field of 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. An expert system is supposed to contain the capability to lead to some conclusion, based on the inputs provided, the system already contains some pre-existing information; which is processed to infer some conclusion. The expert system belongs to the branch of Computer Science called Artificial Intelligence.

Taking into consideration all the points, discussed above, one of the many possible definitions of an Expert System is *: "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."*

Whereas, *the Artificial Intelligence* programs written to achieve expert-level competence in solving problems of 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 as synonyms.** Together they represent the most widespread type of *AI* application.

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–**some things 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.*

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 predicate logic, semantic nets, frames, scripts and rule based systems, exists; and we will discuss them in this unit. Some popular knowledge representation schemes are:

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

As predicate logic have been discussed in previous blocks so we will discuss the remaining knowledge representation schemes here in this unit.

6.1 OBJECTIVES

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

- Understand the basics of expert system
- Understand the basics of Knowledge based systems
- discuss the various knowledge representation scheme like rule based systems, semantic nets, frames, and scripts

6.2 RULE BASED SYSTEMS

We know 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.

In order to plan a solution to the problem, one should have the knowledge of the nature and the structure of the problem domain, 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.

Let's begin with the 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**,
- ***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.

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.

6.2.1 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.

6.2.2 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**.

6.2.3 Conflict Resolution

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

Rule-based systems vary greatly in their details and syntax, 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.

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.

Some of the <u>conflict resolution strategies</u> 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.

To understand, 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.

Check your Progress - 1

Exercise 1 ; In the "Animal Identifier System" discussed above use forward chaining to try to identify the animal called "raja".

6.3 SEMANTIC NETS

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 1.1*.

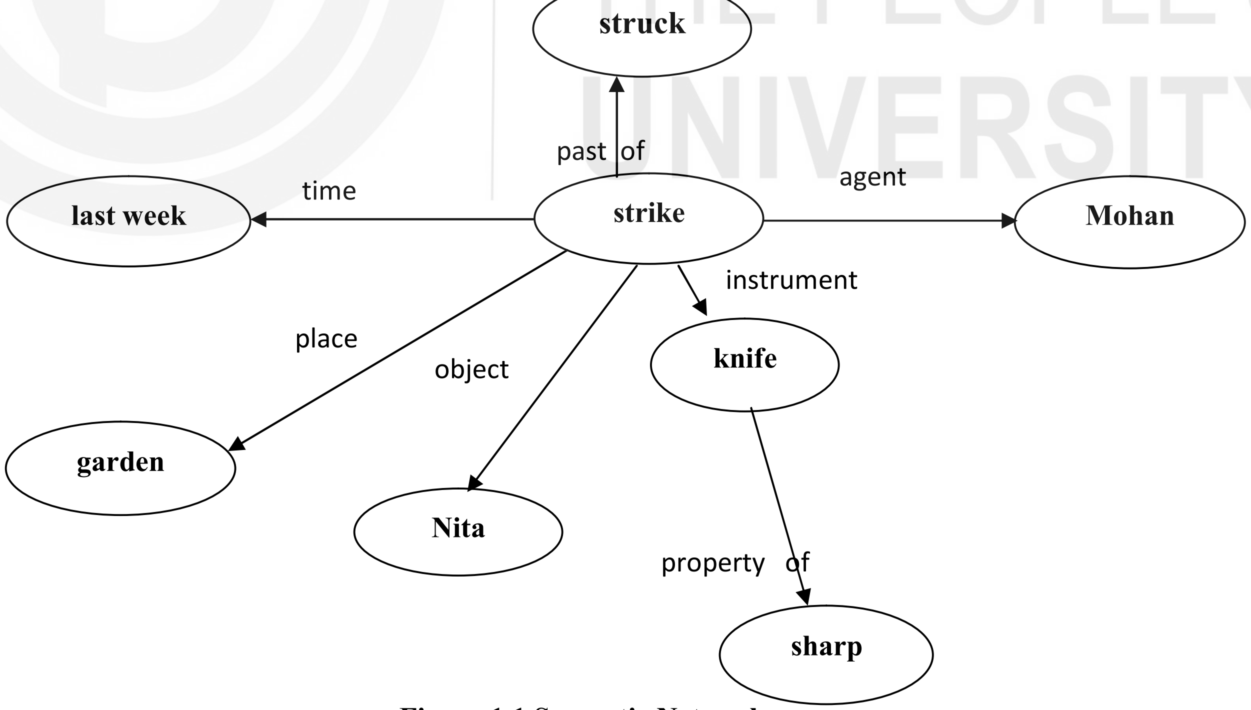

Figure 1.1 Semantic Network

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.

Check Your Progress – 2
Exercise 2: Draw a semantic network for the following English statement:
Mohan struck Nita and Nita's mother struck Mohan.

6.4 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.

Check Your Progress – 3
Exercise 3: 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.

6.5 SCRIPTS

A script is a structured representation describing a stereotyped sequence of events in a particular context.

Scripts are used in natural language understanding systems to organize a knowledge base in terms of the situations that the system should understand. Scripts use a frame-like structure to represent the commonly occurring experience like going to the movies eating in a restaurant, shopping in a supermarket, or visiting an ophthalmologist.

Thus, a script is a structure that prescribes a set of circumstances that could be expected to follow on from one another.

Scripts are beneficial because:

- Events tend to occur in known runs or patterns.
- A casual relationship between events exist.
- An entry condition exists which allows an event to take place.
- Prerequisites exist upon events taking place.

Components of a script

The components of a script include:

- **Entry condition:** These are basic condition which must be fulfilled before events in the script can occur.
- **Results:** Condition that will be true after events in script occurred.
- **Props:** Slots representing objects involved in events
- **Roles:** These are the actions that the individual participants perform.
- **Track:** Variations on the script. Different tracks may share components of the same scripts.
- **Scenes:** The sequence of events that occur.

Describing a script, special symbols of actions are used. These are:

Symbol	Meaning	Example
ATRANS	transfer a relationship	give
PTRANS	transfer physical location of an object	go
PROPEL	apply physical force to an object	push
MOVE	move body part by owner	kick
GRASP	grab an object by an actor	hold
INGEST	taking an object by an animal eat	drink
EXPEL	expel from animal's body	cry
MTRANS	transfer mental information	tell
MBUILD	mentally make new information	decide
CONC	conceptualize or think about an idea	think
SPEAK	produce sound	Say
ATTEND	focus sense organ	listen

Example:-Script for going to the bank to withdraw money.

SCRIPT : Withdraw money

TRACK : Bank

PROPS : Money

Counter

Form

Token

Roles :

P= Customer

E= Employee

C= Cashier

Entry conditions: P has no or less money.

The bank is open.

Results : P has more money.

Scene 1: Entering

P PTRANS P into the Bank

P ATTEND eyes to E

P MOVE P to E

Scene 2: Filling form

P MTRANS signal to E

E ATRANS form to P

P PROPEL form for writing

P ATRANS form to P

ATRANS form to P

Scene 3: Withdrawing money

P ATTEND eyes to counter

P PTRANS P to queue at the counter

P PTRANS token to C

C ATRANS money to P

Scene 4: Exiting the bank

P PTRANS P to out of bank

Advantages of Scripts

- Ability to predict events.
- A single coherent interpretation maybe builds up from a collection of observations.

Disadvantages of Scripts

- Less general than frames.
- May not be suitable to represent all kinds of knowledge

6.6 SUMMARY

This unit majorly discussed the various knowledge representation mechanisms, used in Artificial Intelligence. The unit begins with the discussion on Rule Based Systems, and discussed the related concept of Forward chaining and Backward chaining, later the concept of Conflict resolution is discussed. The unit also discussed the other techniques of knowledge representation like Semantic nets, Frames and Scripts; along with relevant examples for each.

6.7 SOLUTIONS/ANSWERS

Check Your Progress – 1

Exercise 1: Refer to section 6.2

Check Your Progress – 2

Exercise 2: Refer to section 6.3

Check Your Progress – 3

Exercise 3: Refer to section 6.4

6.8 FURTHER READINGS

1. Ela Kumar, “ Artificial Intelligence”, IK International Publications
2. E. Rich and K. Knight, “Artificial intelligence”, Tata Mc Graw Hill Publications
3. N.J. Nilsson, “Principles of AI”, Narosa Publ. House Publications
4. John J. Craig, “Introduction to Robotics”, Addison Wesley publication
5. D.W. Patterson, “Introduction to AI and Expert Systems" Pearson publication