**AIMAN COLLEGE OF ARTS & SCIENCE FOR WOMEN,TIRUCHIRAPPALLI**

**Class : I M.Sc Computer Science – II Semester**

**Subject Title : Artificial Intelligence**

**Subject Code : P16CSE2B**

**Staff Name : D. Lena Vino**

**Designation : Asst. Professor**

#### What is Artificial Intelligence?

Artificial Intelligence (AI) is a branch of *Science* which deals with helping machines finding solutions to complex problems in a more human-like fashion. This generally involves borrowing characteristics from human intelligence, and applying them as algorithms in a computer friendly way. A more or less flexible or efficient approach can be taken depending on the requirements established, which influences how artificial the intelligent behaviour appears. AI is generally associated with *Computer Science*, but it has many important links with other fields such as *Maths*, *Psychology*, *Cognition*, *Biology* and *Philosophy*, among many others. Our ability to combine knowledge from all these fields will ultimately benefit our progress in the quest of creating an intelligent artificial being.

AI currently encompasses a huge variety of subfields, from general-purpose areas such as perception and logical reasoning, to specific tasks such as playing chess, proving mathematical theorems, writing poetry, and diagnosing diseases. Often, scientists in other fields move gradually into artificial intelligence, where they find the tools and vocabulary to systematize and automate the intellectual tasks on which they have been working all their lives. Similarly, workers in AI can choose to apply their methods to any area of human intellectual endeavour. In this sense, it is truly a universal field.

**Definition of Artificial intelligence**

1. AI is the study of how to make computers do things which at the moment people do better. This is ephemeral as it refers to the current state of computer science and it excludes a major area ; problems that cannot be solved well either by computers or by people at the moment.
2. AI is a field of study that encompasses computational techniques for performing tasks that apparently require intelligence when performed by humans.
3. AI is the branch of computer science that is concerned with the automation of intelligent behaviour. A I is based upon the principles of computer science namely data structures used in knowledge representation, the algorithms needed to apply that knowledge and the languages and programming techniques used in their implementation.
4. AI is the field of study that seeks to explain and emulate intelligent behaviour in terms of computational processes.
5. AI is about generating representations and procedures that automatically or autonomously solve problems heretofore solved by humans.
6. A I is the part of computer science concerned with designing intelligent computer systems, that is, computer systems that exhibit the characteristics we associate with intelligence in human behaviour such as understanding language, learning, reasoning and solving problems.
7. A I is the study of mental faculties through the use of computational models.
8. A I is the study of the computations that make it possible to perceive, reason, and act.
9. A I is the exciting new effort to make computers think *machines with minds*, in the full and literal sense.
10. AI is concerned with developing computer systems that can store knowledge and effectively use the knowledge to help solve problems and accomplish tasks. This brief statement sounds a lot like one of the commonly accepted goals in the education of humans. We want students to learn (gain knowledge) and to learn to use this knowledge to help solve problems and accomplish tasks.

**WEAK AND STRONG AI**

There are two conceptual thoughts about AI namely the Weak AI and Strong AI. The strong AI is very much promising about the fact that the machine is almost capable of solve a complex problem like an intelligent man. They claim that a computer is much more efficient to solve the problems than some of the human experts. According to strong AI, the computer is not merely a tool in the study of mind, rather the appropriately programmed computer is really a mind. Strong AI is the supposition that some forms of artificial intelligence can truly reason and solve problems. The term strong AI was originally coined by John Searle.

In contrast, the weak AI is not so enthusiastic about the outcomes of AI and it simply says that some thinking like features can be added to computers to make them more useful tools. It says that computers to make them more useful tools. It says that computers cannot be made intelligent equal to human being, unless constructed significantly differently. They claim that computers may be similar to human experts but not equal in any cases. Generally weak AI refers to the use of software to study or accomplish specific problem solving that do not encompass the full range of human cognitive abilities. An example of weak AI would be a chess program. Weak AI programs cannot be called “intelligent” because they cannot really think.

## TASK DOMAIN OF AI

#### Areas of Artificial Intelligence

* **Perception**
  + **Machine Vision:** It is easy to interface a TV camera to a computer and get an image into memory; the problem is *understanding*what the image represents. Vision takes *lots* of computation; in humans, roughly 10% of all calories consumed are burned in vision computation.
  + **Speech Understanding:** Speech understanding is available now. Some systems must be trained for the individual user and require pauses between words. Understanding continuous speech with a larger vocabulary is harder.
  + **Touch(*tactile* or *haptic*) Sensation:** Important for robot assembly tasks.
* **Robotics** Although industrial robots have been expensive, robot hardware can be cheap: Radio Shack has sold a working robot arm and hand for $15. The limiting factor in application of robotics is not the cost of the robot hardware itself. What is needed is perception and intelligence to tell the robot what to do; ``blind'' robots are limited to very well-structured tasks (like spray painting car bodies).
* **Planning** Planning attempts to order actions to achieve goals. Planning applications include logistics, manufacturing scheduling, planning manufacturing steps to construct a desired product. There are huge amounts of money to be saved through better planning.
* **Expert Systems** Expert Systems attempt to capture the knowledge of a human expert and make it available through a computer program. There have been many successful and economically valuable applications of expert systems. Expert systems provide the following benefits
* Reducing skill level needed to operate complex devices.
* Diagnostic advice for device repair.
* Interpretation of complex data.
* ``Cloning'' of scarce expertise.
* Capturing knowledge of expert who is about to retire.
* Combining knowledge of multiple experts.
* **Theorem Proving** Proving mathematical theorems might seem to be mainly of academic interest. However, many practical problems can be cast in terms of theorems. A general theorem prover can therefore be widely applicable.

**Examples:**

* Automatic construction of compiler code generators from a description of a CPU's instruction set.
* J Moore and colleagues proved correctness of the floating-point division algorithm on AMD CPU chip.
* **Symbolic Mathematics** Symbolic mathematics refers to manipulation of *formulas*, rather than arithmetic on numeric values.
* Algebra
* Differential and Integral Calculus

Symbolic manipulation is often used in conjunction with ordinary scientific computation as a generator of programs used to actually do the calculations. Symbolic manipulation programs are an important component of scientific and engineering workstations.

* **Game Playing** Games are good vehicles for research because they are well formalized, small, and self-contained. They are therefore easily programmed. Games can be good models of competitive situations, so principles discovered in game-playing programs may be applicable to practical problems.

**AI Technique**

Intelligence requires knowledge but knowledge possesses less desirable properties such as

* It is voluminous
* it is difficult to characterise accurately
* it is constantly changing
* it differs from data by being organised in a way that corresponds to its application

An AI technique is a method that exploits knowledge that is represented so that

* The knowledge captures generalisations; situations that share properties, are grouped together, rather than being allowed separate representation.
* It can be understood by people who must provide it; although for many programs the bulk of the data may come automatically, such as from readings. In many AI domains people must supply the knowledge to programs in a form the people understand and in a form that is acceptable to the program.
* It can be easily modified to correct errors and reflect changes in real conditions.
* It can be widely used even if it is incomplete or inaccurate.
* It can be used to help overcome its own sheer bulk by helping to narrow the range of possibilities that must be usually considered.

**Problem Spaces and Search**

Building a system to solve a problem requires the following steps

* Define the problem precisely including detailed specifications and what constitutes an acceptable solution;
* Analyse the problem thoroughly for some features may have a dominant affect on the chosen method of solution;
* Isolate and represent the background knowledge needed in the solution of the problem;
* Choose the best problem solving techniques in the solution.

**Defining the Problem as state Search**

To understand what exactly artificial intelligence is, we illustrate some common problems. Problems dealt with in artificial intelligence generally use a common term called 'state'. A state represents a status of the solution at a given step of the problem solving procedure. The solution of a problem, thus, is a collection of the problem states. The problem solving procedure applies an operator to a state to get the next state. Then it applies another operator to the resulting state to derive a new state. The process of applying an operator to a state and its subsequent transition to the next state, thus, is continued until the goal (desired) state is derived. Such a method of solving a problem is generally referred to as state space approach For example, in order to solve the problem play a game, which is restricted to two person table or board games, we require the rules of the game and the targets for winning as well as a means of representing positions in the game. The opening position can be defined as the initial state and a winning position as a goal state, there can be more than one. legal moves allow for transfer from initial state to other states leading to the goal state. However the rules are far too copious in most games especially chess where they exceed the number of particles in the universe 10. Thus the rules cannot in general be supplied accurately and computer programs cannot easily handle them. The storage also presents another problem but searching can be achieved by hashing. The number of rules that are used must be minimised and the set can be produced by expressing each rule in as general a form as possible. The representation of games in this way leads to a state space representation and it is natural for well organised games with some structure. This representation allows for the formal definition of a problem which necessitates the movement from a set of initial positions to one of a set of target positions. It means that the solution involves using known techniques and a systematic search. This is quite a common method in AI.

**Formal description of a problem**

* Define a state space that contains all possible configurations of the relevant objects, without enumerating all the states in it. A *state space* represents a problem in terms of *states* and *operators* that change states
* Define some of these states as possible initial states;
* Specify one or more as acceptable solutions, these are goal states;
* Specify a set of rules as the possible actions allowed. This involves thinking about the generality of the rules, the assumptions made in the informal presentation and how much work can be anticipated by inclusion in the rules.

The control strategy is again not fully discussed but the AI program needs a structure to facilitate the search which is a characteristic of this type of program.

**Example:**

**The water jug problem :**There are two jugs called **four** and **three** ; four holds a maximum of four gallons and **three** a maximum of three gallons. How can we get 2 gallons in the jug **four**. The state space is a set of ordered pairs giving the number of gallons in the pair of jugs at any time ie (**four, three**) where **fou**r = 0, 1, 2, 3, 4 and **three** = 0, 1, 2, 3. The start state is (0,0) and the goal state is (2,n) where n is a don't care but is limited to **three** holding from 0 to 3 gallons. The major production rules for solving this problem are shown below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Initial** | **condition** | **goal** | **comment** |
| 1 (four,three) | if four < 4 | (4,three) | fill four from tap |
| 2 (four,three) | if three< 3 | (four,3) | fill three from tap |
| 3 (four,three) | If four > 0 | (0,three) | empty four into drain |
| 4 (four,three) | if three > 0 | (four,0) | empty three into drain |
| 5 (four,three) if four+three<4 | | (four+three,0) | empty three into four |
| 6 (four,three) if four+three<3 | | (0,four+three) | empty four into three |
| 7 (0,three) If three>0 | | (three,0) | empty three into four |
| 8 (four,0) if four>0 | | (0,four) | empty four into three |
| 9 (0,2) | | (2,0) | empty three into four |
| 10 (2,0) | | (0,2) | empty four into three |
| 11 (four,three) if four<4 | | (4,three-diff) | pour diff, 4-four, into four from three |

12 (three,four) if three<3 (four-diff,3) pour diff, 3-three, into three from four and a solution is given below Jug four, jug three rule applied

0 0

0 3 2

3 0 7

3 3 2

4 2 11

0 2 3

2 0 10

**Control strategies.**

A good control strategy should have the following requirement: The first requirement is that it causes motion. In a game playing program the pieces move on the board and in the water jug problem water is used to fill jugs. The second requirement is that it is systematic, this is a clear requirement for it would not be sensible to fill a jug and empty it repeatedly nor in a game would it be advisable to move a piece round and round the board in a cyclic way. We shall initially consider two systematic approaches to searching.

## Monotonic and Non monotonic Learning :

**Monotonic learning** is when an agent may not learn any knowledge that contradicts what it already knows. For example, it may not replace a statement with its negation. Thus, the knowledge base may only grow with new facts in a monotonic fashion. The advantages of monotonic learning are:

1.greatly simplified truth-maintenance 2.greater choice in learning strategies

**Non-monotonic learning** is when an agent may learn knowledge that contradicts what it already knows. So it may replace old knowledge with new if it believes there is sufficient reason to do so. The advantages of non-monotonic learning are:

1. increased applicability to real domains,
2. greater freedom in the order things are learned in

A related property is the consistency of the knowledge. If an architecture must maintain a consistent knowledge base then any learning strategy it uses must be monotonic.

## PROBLEM CHARACTERISTICS

A problem may have different aspects of representation and explanation. In order to choose the most appropriate method for a particular problem, it is necessary to analyze the problem along several key dimensions. Some of the main key features of a problem are given below.

 Is the problem decomposable into set of sub problems?

 Can the solution step be ignored or undone?

 Is the problem universally predictable?

 Is a good solution to the problem obvious without comparison to all the possible solutions?

 Is the desire solution a state of world or a path to a state?

 Is a large amount of knowledge absolutely required to solve the problem?

 Will the solution of the problem required interaction between the computer and the person?

The above characteristics of a problem are called as 7-problem characteristics under which the solution must take place.

## PRODUCTION SYSTEM AND ITS CHARACTERISTICS

The production system is a model of computation that can be applied to implement search algorithms and model human problem solving. Such problem solving knowledge can be packed up in the form of little quanta called productions. A production is a rule consisting of a situation recognition part and an action part. A production is a situation-action pair in which the left side is a list of things to watch for and the right side is a list of things to do so. When productions are used in deductive systems, the situation that trigger productions are specified combination of facts. The actions are restricted to being assertion of new facts deduced directly from the triggering combination. Production systems may be called premise conclusion pairs rather than situation action pair.

A production system consists of following components.

* 1. A set of production rules, which are of the form AB. Each rule consists of left hand side constituent that represent the current problem state and a right hand side that represent an output state. A rule is applicable if its left hand side matches with the current problem state.
  2. A database, which contains all the appropriate information for the particular task. Some part of the database may be permanent while some part of this may pertain only to the solution of the current problem.
  3. A control strategy that specifies order in which the rules will be compared to the database of rules and a way of resolving the conflicts that arise when several rules match simultaneously.
  4. A rule applier, which checks the capability of rule by matching the content state with the left hand side of the rule and finds the appropriate rule from database of rules.

The important roles played by production systems include a powerful knowledge representation scheme. A production system not only represents knowledge but also action. It acts as a bridge between AI and expert systems. Production system provides a language in which the representation of expert knowledge is very natural. We can represent knowledge in a production system as a set of rules of the form

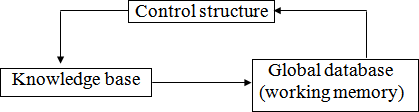
If (condition) THEN (condition)

along with a control system and a database. The control system serves as a rule interpreter and sequencer. The database acts as a context buffer, which records the conditions evaluated by the rules and information on which the rules act. The production rules are also known as condition – action, antecedent – consequent, pattern – action, situation – response, feedback – result pairs.

For example,

If (you have an exam tomorrow) THEN (study the whole night)

The production system can be classified as monotonic, non-monotonic, partially commutative and commutative.



##### Figure Architecture of Production System

**Features of Production System**

Some of the main features of production system are:

**Expressiveness and intuitiveness**: In real world, many times situation comes like “if this happen-you will do that”, “if this is so-then this should happen” and many more. The production rules essentially tell us what to do in a given situation.

1. **Simplicity:** The structure of each sentence in a production system is unique and uniform as they use “IF-THEN” structure. This structure provides simplicity in knowledge representation. This feature of production system improves the readability of production rules.
2. **Modularity:** This means production rule code the knowledge available in discrete pieces. Information can be treated as a collection of independent facts which may be added or deleted from the system with essentially no deletetious side effects.
3. **Modifiability:** This means the facility of modifying rules. It allows the development of production rules in a skeletal form first and then it is accurate to suit a specific application.
4. **Knowledge intensive:** The knowledge base of production system stores pure knowledge. This part does not contain any type of control or programming information. Each production rule is normally written as an English sentence; the problem of semantics is solved by the very structure of the representation.

**Disadvantages of production system**

1. **Opacity:** This problem is generated by the combination ofproduction rules. The opacity is generated because of less prioritization of rules. More priority to a rule has the less opacity.
2. **Inefficiency:** During execution of a program several rules may active. A well devised control strategy reduces this problem. As the rules of the production system are large in number and they are hardly written in hierarchical manner, it requires some forms of complex search through all the production rules for each cycle of control program.
3. **Absence of learning:** Rule based production systems do not store the result of the problem for future use. Hence, it does not exhibit any type of learning capabilities. So for each time for a particular problem, some new solutions may come.
4. **Conflict resolution:** The rules in a production system should not have any type of conflict operations. When a new rule is added to a database, it should ensure that it does not have any conflicts with the existing rules.

## ALGORITHM OF PROBLEM SOLVING

Any one algorithm for a particular problem is not applicable over all types of problems in a variety of situations. So there should be a general problem solving algorithm, which may work for different strategies of different problems.

## Algorithm (problem name and specification)

##### Step 1:

Analyze the problem to get the starting state and goal state.

##### Step 2:

Find out the data about the starting state, goalstate

##### Step 3:

Find out the production rules from initial database for proceeding the problem to goal state.

##### Step 4:

Select some rules from the set of rules that can be applied to data.

##### Step 5:

Apply those rules to the initial state and proceed to get the next state.

##### Step 6:

Determine some new generated states after applying the rules. Accordingly make them as current state.

##### Step 7:

Finally, achieve some information about the goal state from the recently used current state and get the goal state.

##### Step 8:

Exit.

After applying the above rules an user may get the solution of the problem from a given state to another state. Let us take few examples.

## VARIOUS TYPES OF PROBLEMS AND THEIR SOLUTIONS

### Water Jug Problem

#### Definition:

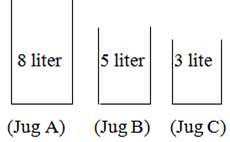
Some jugs are given which should have non-calibrated properties. At least any one of the jugs should have filled with water. Then the process through which we can divide the whole water into different jugs according to the question can be called as water jug problem.

##### Procedure:

Suppose that you are given 3 jugs A,B,C with capacities 8,5 and 3 liters respectively but are not calibrated (i.e. no measuring mark will be there). Jug A is filled with 8 liters of water. By a series of pouring back and forth among the 3 jugs, divide the 8 liters into 2 equal parts i.e. 4 liters in jug A and 4 liters in jug B. How?

In this problem, the start state is that the jug A will contain 8 liters water whereas jug B and jug C will be empty. The production rules involve filling a jug with some amount of water, taking from the jug A. The search will be finding the sequence of production rules which transform the initial state to final state. The state space for this problem can be described by set of ordered pairs of three variables (A, B, C) where variable A represents the 8 liter jug, variable B represents the 5 liter and variable C represents the 3 liters jug respectively.

##### Figure



The production rules are formulated as follows:

##### Step 1:

In this step, the initial state will be (8, 0, 0) as the jug B and jug C will be empty. So the water of jug A can be poured like:

(5, 0, 3) means 3 liters to jug C and 5 liters will remain in jug A. (3, 5, 0) means 5 liters to jug B and 3 liters will be in jug A.

(0, 5, 3) means 5 liters to jug B and 3 liters to jug C and jug C and jug A will be empty.



##### Step2:

In this step, start with the first current state of step-1 i.e. (5, 0, 3). This state can only be implemented by pouring the 3 liters water of jug C into jug B. so the state will be (5, 3, 0). Next, come to the second

current state of step-1 i.e. (3, 5, 0). This state can be implemented by only pouring the 5 liters water of jug B into jug C. So the remaining water in jug B will be 2 liters. So the state will be (3, 2, 3). Finally come to the third current state of step-1 i.e. (0, 5, 3). But from this state no more state can be implemented because after implementing we may get (5, 0, 3) or (3, 5, 0) or (8, 0, 0) which are repeated state. Hence these states are not considerably again for going towards goal.

So the state will be like:

(5, 0, 3) ‹ (5, 3, 0)

(3, 5, 0) ‹ (3, 2, 3)

(0, 5, 3) ‹ X

##### Step 3:

In this step, start with the first current state of step-2 i.e. (5, 3, 0) and proceed likewise the above steps.

(5, 3, 0) ‹ (2, 3, 3)

(3, 2, 3) ‹ (6, 2, 0)

##### Step 4:

In this step, start with the first current state of step-3 i.e. (2, 3, 3) and proceed.

(2, 3, 3) ‹ (2, 5, 1)

(6, 2, 0) ‹ (7, 0, 1)

##### Step 5:

(2, 5, 1) ‹ (7, 0, 1)

(6, 0, 2) ‹ (1, 5, 2)

##### Step6:

(7, 0, 1) ‹ (7, 1, 0)

(1, 4, 3) ‹ (1, 4, 3)

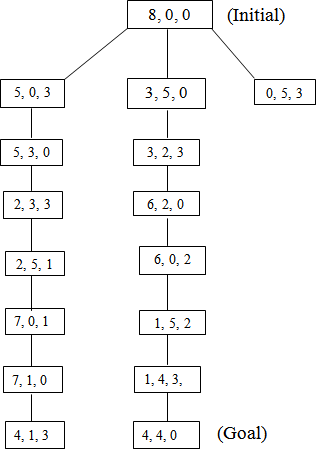
##### Step7:

(7, 1, 0) ‹ (4, 1, 3)

(1, 4, 3) ‹ (4, 4, 0) (Goal)

So finally the state will be (4, 4, 0) that means jug A and jug B contains 4 liters of water each which is our goal state. One thing you have to very careful about the pouring of water from one jug to another that the capacity of jug must satisfy the condition to contain that much of water.

The tree of the water jug problem can be like:



##### Figure Comments:

 This problem takes a lot of time to find the goal state.

 This process of searching in this problem is very lengthy.

 At each step of the problem the user have to strictly follow the production rules. Otherwise the problem may go to infinity step.

### Missionaries and Carnivals Problem

##### Definition:

In Missionaries and Carnivals Problem, initially there are some missionaries and some carnivals will be at a sideof a river. They want to cross the river. But there is only one boat available to cross the river. The capacity of the boat is 2 and no one missionary or no Carnivals can cross the river together. So for solving the problem and to find out the solution on different states is called the Missionaries and Carnival Problem.

##### Procedure:

Let us take an example. Initially a boatman, Grass, Tiger and Goat is present at the left bank of the river and want to cross it. The only boat available is one capable of carrying 2 objects of portions at a time. The condition of safe crossing is that at no time the tiger present with goat, the goat present with the grass at the either side of the river. How they will cross the river?

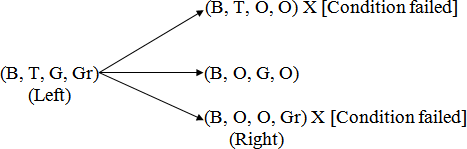
The objective of the solution is to find the sequence of their transfer from one bank of the river to the other using the boat sailing through the river satisfying these constraints.

Let us use different representations for each of the missionaries and Carnivals as follows.

B: Boat T: Tiger G: Goat Gr: Grass

##### Step 1:

According to the question, this step will be (B, T, G, Gr) as all the Missionaries and the Carnivals are at one side of the bank of the river. Different states from this state can be implemented as



`The states (B, T, O, O) and (B, O, O, Gr) will not be countable because at a time the Boatman and the Tiger or the Boatman and grass cannot go. (According to the question).

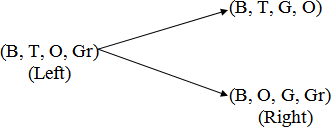
##### Step 2:

Now consider the current state of step-1 i.e. the state (B, O, G, O). The state is the right side of the river. So on the left side the state may be (B, T, O, Gr)

i.e.(B, O, G, O) ‹ (B, T, O, Gr)

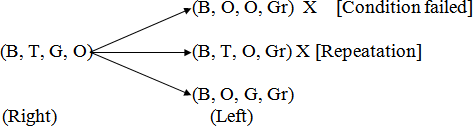
(Right) (Left)

##### Step 3:

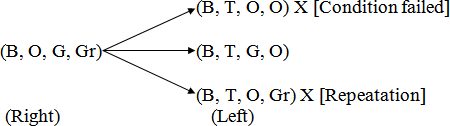
Now proceed according to the left and right sides of the river such that the condition of the problem must be satisfied.

##### Step 4:

First, consider the first current state on the right side of step 3 i.e.

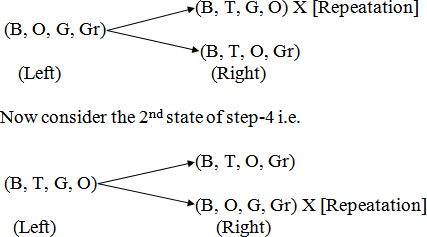


Now consider the second current state on the right side of step-3 i.e.



##### Step 5:

Now first consider the first current state of step 4 i.e.



##### Step 6:



**Step 7:**



Hence the final state will be (B, T, G, Gr) which are on the right side of the river.

##### Comments:

 This problem requires a lot of space for its state implementation.

 It takes a lot of time to search the goal node.

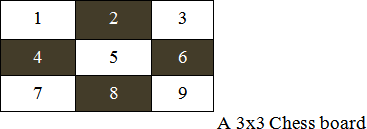
 The production rules at each level of state are very strict.

### Chess Problem

##### Definition:

It is a normal chess game. In a chess problem, the start is the initial configuration of chessboard. The final state is the any board configuration, which is a winning position for any player. There may be multiple final positions and each board configuration can be thought of as representing a state of the game. Whenever any player moves any piece, it leads to different state of game.

##### Procedure:



**Figure**

The above figure shows a 3x3 chessboard with each square labeled with integers 1 to 9. We simply enumerate the alternative moves rather than developing a general move operator because of the reduced size of the problem. Using a predicate called move in predicate calculus, whose parameters are the starting and ending squares, we have described the legal moves on the board. For example, move (1, 8) takes the knight from the upper left-hand corner to the middle of the bottom row. While playing Chess, a knight can move two squares either horizontally or vertically followed by one square in an orthogonal direction as long as it does not move off the board.

The all possible moves of figure are as follows.

Move (1, 8) move (6, 1)

Move (1, 6) move (6, 7)

Move (2, 9) move (7, 2)

Move (2, 7) move (7, 6)

Move (3, 4) move (8, 3)

Move (3, 8) move (8, 1)

Move (4, 1) move (9, 2)

Move (4, 3) move (9, 4)

The above predicates of the Chess Problem form the knowledge base for this problem. An unification algorithm is used to access the knowledge base.

Suppose we need to find the positions to which the knight can move from a particular location, square 2. The goal move (z, x) unifies with two different predicates in the knowledge base, with the substitutions

{7/x} and {9/x}. Given the goal move (2, 3), the responsible is failure, because no move (2, 3) exists in the knowledge base.

##### Comments:

 In this game a lots of production rules are applied for each move of the square on the chessboard.

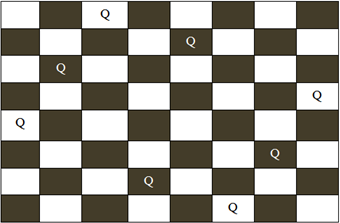
 A lots of searching are required in this game.

 Implementation of algorithm in the knowledge base is very important.

### Queen Problem

##### Definition:

“We have 8 queens and an 8x8 Chess board having alternate black and white squares. The queens are placed on the chessboard. Any queen can attack any other queen placed on same row, or column or diagonal. We have to find the proper placement of queens on the Chess board in such a way that no queen attacks other queen”.



##### Procedure:

**Figure A possible board configuration of 8 queen problem**

In figure , the possible board configuration for 8-queen problem has been shown. The board has alternative black and white positions on it. The different positions on the board hold the queens. The production rule for this game is you cannot put the same queens in a same row or same column or in same diagonal. After shifting a single queen from its position on the board, the user have to shift other queens according to the production rule. Starting from the first row on the board the queen of their corresponding row and column are to be moved from their original positions to another position. Finally the player has to be ensured that no rows or columns or diagonals of on the table is same.

##### Comments:

 This problem requires a lot of space to store the board.

 It requires a lot of searching to reach at the goal state.

 The time factor for each queen’s move is very lengthy.

 The problem is very strict towards the production rules.

### 8- Puzzle Problem

##### Definition:

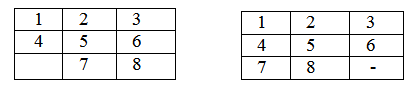
“It has set off a 3x3 board having 9 block spaces out of which 8 blocks having tiles bearing number from 1 to 8. One space is left blank. The tile adjacent to blank space can move into it. We have to arrange the tiles in a sequence for getting the goal state”.

##### Procedure:

The 8-puzzle problem belongs to the category of “sliding block puzzle” type of problem. The 8-puzzle is a square tray in which eight square tiles are placed. The remaining ninth square is uncovered. Each tile in the tray has a number on it. A tile that is adjacent to blank space can be slide into that space. The game consists of a starting position and a specified goal position. The goal is to transform the starting position into the goal position by sliding the tiles around. The control mechanisms for an 8-puzzle solver must keep track of the order in which operations are performed, so that the operations can be undone one at a time if necessary. The objective of the puzzles is to find a sequence of tile movements that leads from a starting configuration to a goal configuration such as two situations given below.

##### Figure (Starting State) (Goal State)

The state of 8-puzzle is the different permutation of tiles within the frame. The operations are the permissible moves up, down, left, right. Here at each step of the problem a function f(x) will be defined



which is the combination of g(x) and h(x). i.e.

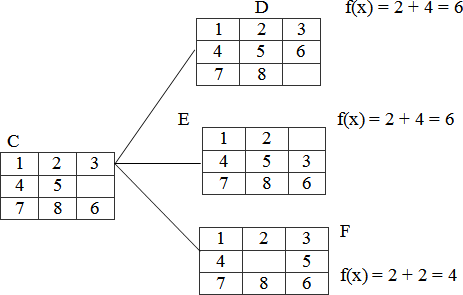
Where

F(x)=g(x) + h (x)

g (x): how many steps in the problem you have already done or the current state from the initial state.

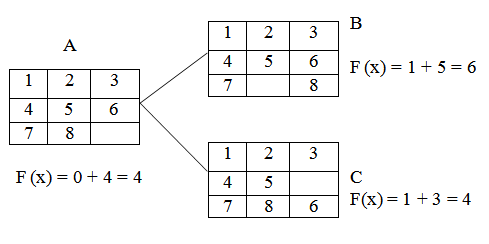
h (x): Number of ways through which you can reach at the goal state from the current state or Or

h (x)is the heuristic estimator that compares the current state with the goal state note down how many states are displaced from the initial or the current state. After calculating the f (x) value at each step finally take the smallest f (x) value at every step and choose that as the next current state to get the goal state.



Let us take an example.

##### Figure (Initial State) (Goal State)



**Step1:**

f (x)is the step required to reach at the goal state from the initial state. So in the tray either 6 or 8 can change their portions to fill the empty position. So there will be two possible current states namely B and

C. The f (x) value of B is 6 and that of C is 4. As 4 is the minimum, so take C as the current state to the next state.

##### Step 2:

In this step, from the tray C three states can be drawn. The empty position will contain either 5 or 3 or 6. So for three different values three different states can be obtained. Then calculate each of their f (x) and

take the minimum one.

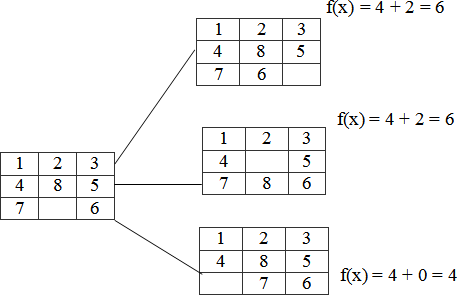
Here the state F has the minimum value i.e. 4 and hence take that as the next current state.

##### Step 3:

The tray F can have 4 different states as the empty positions can be filled with b4 values i.e.2, 4, 5, 8.

##### Step 4:

In the step-3 the tray I has the smallest f (n) value. The tray I can be implemented in 3 different states because the empty position can be filled by the members like 7, 8, 6.



Hence, we reached at the goal state after few changes of tiles in different positions of the trays.

##### Comments:

 This problem requires a lot of space for saving the different trays.

 Time complexity is more than that of other problems.

 The user has to be very careful about the shifting of tiles in the trays.

 Very complex puzzle games can be solved by this technique.

### Monkey Banana Problem

##### Definition:

“A monkey is in a room. A bunch of bananas is hanging from the ceiling. The monkey cannot reach the bananas directly. There is a box in the corner of the room. How can the monkey get the bananas?”

##### Procedure:

The solution of the problem is of course that the monkey must push the box under the bananas, then stand on the box and grab the bananas. But the solution procedure requires a lot of planning algorithms. The purpose of the problem is to raise the question: Are monkeys intelligent? Both humans and monkeys have the ability to use mental maps to remember things like where to go to find shelter or how to avoid danger. They can also remember where to go to gather food and water, as well as how to communicate with each other. Monkeys have the ability not only to remember how to hunt and gather but they also have the ability to learn new things, as is the case with the monkey and the bananas. Even though that monkey may never have entered that room before or had only a box for a tool to gather the food available, that monkey can learn that it needs to move the box across the floor, position it below the bananas and climb the box to reach for them. Some people believe that this is part instinct, part learned behaviour. It is most probably both.

Initially, the monkey is at location ‘A’, the banana is at location ‘B’ and the box is at location ‘C’. The monkey and box have height “low”; but if the monkey climbs onto the box will have height “High”, the same as the bananas.

The action available to the monkey include: “GO” from one place to another.

“PUSH” an object from one place to another. “Climb” onto an object.

“Grasp” an object.

Grasping results in holding the object if the monkey and the object are in the same place at the same height.

The solution of the problem in different steps can be of followings.

1. What is the initial state description? At (monkey, A), At (banana, B), At (box, C) Position (monkey, low), Position (banana, high), Position (box, low)
2. What are the definitions of the different actions?
   1. Go (x, y) Precondition: At (monkey, x) Effects: ¬At (monkey, x), At (monkey, y)
   2. Push (object, x, y, height) Pre condition: At (monkey, x), At (object, x), Position (monkey, height), Position (object, height)

Effects: ¬ (monkey, x), ¬At (object, x), At (monkey, y), At(object, y)\

* 1. Climb up (object, y) Precondition: At (monkey, x), At (object, x), Position (monkey, low), Position (object, low) Effects: ¬Position (monkey, low), Position (monkey, high), On (monkey, object)
  2. Climb down (object) Preconditions: Position (monkey, high), On (monkey, object) Effects: ¬Position (monkey, high), ¬ On (monkey, object) Position: (monkey, low)
  3. Grasp (object, x, height) Preconditions: At (monkey, x), At (object, x), Position (monkey, height), Position (object, height)

Effect: Hold (object)

* 1. UnGrasp (object, x, height) Preconditions: Hold (object), At (monkey, x), At (object, x), Position (monkey, height), Position (object, height) Effects: ¬Hold (object)

So the solution to the planning problem may be of following

GO(A,C)

PUSH (Box, C, B, Low)

Climb Up(Box , B)

Grasp(banana, B, High)

Climb down(Box)

Push(Box, B, C, Low)

##### Comments:

 One major application of the monkey banana problem is the toy problem of computer science.

 One of the specialized purposes of the problem is to raise the question: Are monkeys intelligent?

 This problem is very useful in logic programming and planning.

### Tower of Hanoi Problem

##### Definition:

“We are given a tower of eight discs (initially) four in the applet below, initially stacked in increasing size on one of three pegs. The objective is to transfer the entire tower to one of the other pegs (the right most one in the applet below), moving only one disc at a time and never a larger one onto a smaller”.

##### Procedure:

The tower of Hanoi puzzle was invented by the French mathematician Eduardo Lucas in 1883. The puzzle is well known to students of computer science since it appears in virtually any introductory text on data structure and algorithms.

The objective of the puzzle is to move the entire stack to another rod, obeying the following rules.

 Only one disc can be moved at a time.

 Each move consist of taking the upper disc from one of the rods and sliding it onto another rod, on top of the other discs that may already be present on that rod.

 No disc may be placed on the top of a smaller disk.

There is a legend about a Vietnamese temple which contains a large room with three times. Worn posts in it surrounded by 64 golden disks. The priests of Hanoi, acting out of command of an ancient prophecy, have been moving these disks, in accordance with the rules of the puzzle, since that time. The puzzle is therefore also known as the tower of Brahma puzzle. According to the legend, when the last move of the puzzle is completed, the world will end.

There are many variations on this legend. For instance, in some tellings, the temple is a monastery and the priests are monks. The temple or monastery may be said to be in different parts of the world including Hanoi, Vietnam and may be associated with any religion. The flag tower of Hanoi may have served as the inspiration for the name.

The puzzle can be played with any number of disks, although many toy versions have around seven to nine of them. The game seems impossible to many novices yet is solvable with a simple algorithm. The following solution is a very simple method to solve the tower of Hanoi problem.

 Alternative moves between the smallest piece and a non- smallest piece. When moving the smallest piece, always move it in the same direction (to the right if starting number of pieces is even, to the left if starting number of pieces is odd).

 If there is no tower in the chosen direction, move the pieces to the opposite end, but then continue to move in the correct direction, for example if you started with three pieces, you would move the smallest piece to the opposite end, then continue in the left direction after that.

 When the turn is to move the non-smallest piece, there is only one legal move.

Doing this should complete the puzzle using the least amount of moves to do so. Finally, the user will reach at the goal. Also various types of solutions may be possible to solve the tower of Hanoi problem like recursive procedure, non-recursive procedure and binary solution procedure.

Another simple solution to the problem is given below. For an even number of disks

 Make the legal move between pegs A and B.

 Make the legal moves between pages A and C.

 Make the legal move between pages B and C. For an even number of disks

 Make the legal move between pegs A and C.

 Make the legal move between pegs A and B.

 Make the legal move between pegs B and C.

 Repeat until complete.

A recursive solution for tower of Hanoi problem is as follows.

A key to solving this problem is to recognize that it can be solve by breaking the problem down into the collection of smaller problems and further breaking those problems down into even smaller problems until a solution is reached. The following procedure demonstrates this approach.

 Label the pegs A, B, C - these levels may move at different steps.

 Let n be the total number of disks.

 Number of disks from 1 (smallest, topmost) to n (largest, bottommost).

To move n disks from peg A to peg C.

1. Move n-1 disks from A to B. This leaves disk #n alone on peg A.
2. Move disk #n from A to C.
3. Move n-1 disks from B to C so they sit on disk #n.

To carry out steps a and c, apply the same algorithm again for n-1. The entire procedure is a finite number of steps, since at most point the algorithm will be required for n = 1. This step, moving a single disc from peg A to peg B, is trivial.

##### Comments:

 The tower of Hanoi is frequently used in psychological research on problem solving.

 This problem is frequently used in neuro-psychological diagnosis and treatment of executive functions.

 The tower of Hanoi is also used as backup rotation scheme when performing computer data backups where multiple tabs/media are involved.

 This problem is very popular for teaching recursive algorithm to beginning programming students.

 A pictorial version of this puzzle is programmed into emacs editor, accessed by typing M - X Hanoi.

 The tower of Hanoi is also used as a test by neuro-psychologists trying to evaluate frontal lobe deficits.

### Cryptarithmatic Problem

##### Definition:

“It is an arithmetic problem which is represented in letters. It involves the decoding of digit represented by a character. It is in the form of some arithmetic equation where digits are distinctly represented by some characters. The problem requires finding of the digit represented by each character. Assign a decimal digit to each of the letters in such a way that the answer to the problem is correct. If the same letter occurs more than once, it must be assigned the same digit each time. No two different letters may be assigned the same digit”.

##### Procedure:

Cryptarithmatic problem is an interesting constraint satisfaction problem for which different algorithms have been developed. Cryptarithm is a mathematical puzzle in which digits are replaced by letters of the alphabet or other symbols. Cryptarithmatic is the science and art of creating and solving cryptarithms.

The different constraints of defining a cryptarithmatic problem are as follows.

* 1. Each letter or symbol represented only one and a unique digit throughout the problem.
  2. When the digits replace letters or symbols, the resultant arithmetical operation must be correct. The above two constraints lead to some other restrictions in the problem.

For example:



Consider that, the base of the number is 10. Then there must be at most 10 unique symbols or letters in the problem. Otherwise, it would not possible to assign a unique digit to unique letter or symbol in the problem. To be semantically meaningful, a number must not begin with a 0. So, the letters at the beginning of each number should not correspond to 0. Also one can solve the problem by a simple blind search. But a rule based searching technique can provide the solution in minimum time.

Now, let us solve a simple cryptarithmatic puzzle given below.

##### Step 1:

In the above problem, M must be 1. You can visualize that, this is an addition problem. The sum of two four digit numbers cannot be more than 10,000. Also M cannot be zero according to the rules, since it is



the first letter.

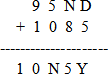
So now you have the problem like



##### Step 2:

Now in the column s10, s+1 ≥ 10. S must be 8 because there is a 1 carried over from the column EON or

1. O must be 0 (if s=8 and there is a 1 carried or s = 9 and there is no 1 carried) or 1 (if s=9 and there is a 1 carried). But 1 is already taken, so O must be 0.





##### Step 3:

There cannot be carry from column EON because any digit +0 < 10, unless there is a carry from the column NRE, and E=9; But this cannot be the case because then N would be 0 and 0 is already taken. So E < 9 and there is no carry from this column. Therefore S=9 because 9+1=10.

##### Step 4:



In the column EON, E cannot be equal to N. So there must be carry from the column NRE; E+1=N. We now look at the column NRE, we know that E+1=N. Since we know that carry from this column, N+R=1E (if there is no carry from the column DEY) or N+R+1=1E (if there is a carry from the column DEY).

Let us see both the cases:

No carry: N + R = 10 + ( N — 1) = N + 9

R = 9

But 9 is already taken, so this will not work Carry: N + R + 1 = 9

R = 9 — 1 = 8 This must be solution for R



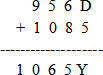
##### Step 5:

Now just think what are the digits we have left? They are 7, 6, 5, 4, 3 and 2. We know there must be a carry from the column DEY. So D + E Σ 10.N = E + 1, So E cannot be 7 because then N would be 8

which is already taken. D is almost 7, so E cannot be 2 because then D + E € 10 and E cannot be 3 because then D + E = 10 and Y = 0, but 0 is already taken. Also E cannot be 4 because if D Σ 6, D + E € 10 and if D = 6 or D = 7 then Y = 0 or Y = 1, which are both taken. So E is 5 or 6. If E = 6, then D = 7 and Y = 3. So this part will work but look the column N8E. Point that there is a carry from the column D5Y.N + 8 + 1 = 16(As there is a carry from this column). But then N=7 and 7 is taken by D therefore E=5.

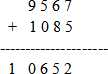
##### Step 6:

Now we have gotten this important digit, it gets much simpler from here. N+8+1=15, N=6



##### Step 7:

The digits left are 7, 4, 3 and 2. We know there is carry from the column D5Y, so the only pair that works is D=7 and Y= 2.



Which is final solution of the problem.

##### Comments:

 This problem requires a lot of reasoning.

 Time complexity of the problem is more as concerned to the other problems.

 This problem can also be solved by the evolutionary approach and mutation operations.

 This problem is dependent upon some constraints which are necessary part of the problem.

 Various complex problems can also be solved by this technique.

## SEARCHING

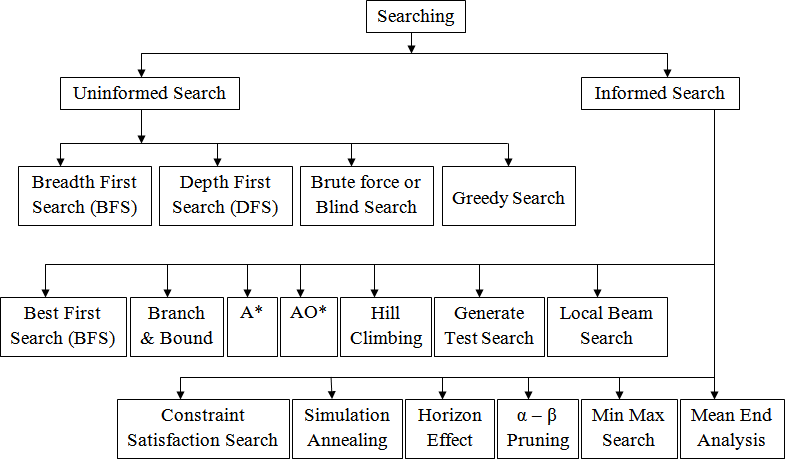
Problem solving in artificial intelligence may be characterized as a systematic search through a range of possible actions in order to reach some predefined goal or solution. In AI problem solving by search

algorithms is quite common technique. In the coming age of AI it will have big impact on the technologies of the robotics and path finding. It is also widely used in travel planning. This chapter contains the different search algorithms of AI used in various applications. Let us look the concepts for visualizing the algorithms.

A search algorithm takes a problem as input and returns the solution in the form of an action sequence. Once the solution is found, the actions it recommends can be carried out. This phase is called as the execution phase. After formulating a goal and problem to solve the agent cells a search procedure to solve it. A problem can be defined by 5 components.

* 1. **The initial state**: The state from which agent will start.
  2. **The goal state**: The state to be finally reached.
  3. **The current state**: The state at which the agent is present after starting from the initial state.
  4. **Successor function**: It is the description of possible actions and their outcomes.
  5. **Path cost**: It is a function that assigns a numeric cost to each path.

## DIFFERENT TYPES OF SEARCHING

the searching algorithms can be various types. When any type of searching is performed, there may some information about the searching or mayn’t be. Also it is possible that the searching procedure may depend upon any constraints or rules. However, generally searching can be classified into two types i.e. uninformed searching and informed searching. Also some other classifications of these searches are given below in the figure .

**UNINFORMED SEARCH**

**Breadth First Search (BFS)**

Breadth first search is a general technique of traversing a graph. Breadth first search may use more memory but will always find the shortest path first. In this type of search the state space is represented in form of a tree. The solution is obtained by traversing through the tree. The nodes of the tree represent the start value or starting state, various intermediate states and the final state. In this search a queue data structure is used and it is level by level traversal. Breadth first search expands nodes in order of their distance from the root. It is a path finding algorithm that is capable of always finding the solution if one exists. The solution which is found is always the optional solution. This task is completed in a very memory intensive manner. Each node in the search tree is expanded in a breadth wise at each level.

## Concept:

**Step 1:** Traverse the root node

**Step 2:** Traverse all neighbours of root node.

**Step 3:** Traverse all neighbours of neighbours of the root node.

**Step 4:** This process will continue until we are getting the goal node.

## Algorithm:

**Step 1:** Place the root node inside the queue.

**Step 2:** If the queue is empty then stops and return failure.

**Step 3:** If the FRONT node of the queue is a goal node then stop and return success.

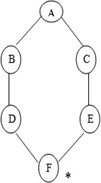
**Step 4:** Remove the FRONT node from the queue. Process it and find all its neighbours that are in ready state then place them inside the queue in any order.

**Step 5:** Go to Step 3.

**Step 6:** Exit.

## Implementation:

Let us implement the above algorithm of BFS by taking the following suitable example.



##### Figure

Consider the graph in which let us take A as the starting node and F as the goal node (\*)

##### Step 1:

Place the root node inside the queue i.e. A

A

##### Step 2:

Now the queue is not empty and also the FRONT node i.e. A is not our goal node. So move to step 3.

##### Step 3:

So remove the FRONT node from the queue i.e. A and find the neighbour of A i.e. B and C

A

C

B

##### Step 4:

Now b is the FRONT node of the queue .So process B and finds the neighbours of B i.e. D.

B

C D

##### Step 5:

Now find out the neighbours of C i.e. E

C

E

D

##### Step 6:

Next find out the neighbours of D as D is the FRONT node of the queue

D

##### Step 7:

F

E

Now E is the front node of the queue. So the neighbour of E is F which is our goal node.

F

E

##### Step 8:

Finally F is our goal node which is the FRONT of the queue. So exit.

F

## Advantages:

 In this procedure at any way it will find the goal.

 It does not follow a single unfruitful path for a long time.

 It finds the minimal solution in case of multiple paths.

## Disadvantages:

 BFS consumes large memory space.

 Its time complexity is more.

 It has long pathways, when all paths to a destination are on approximately the same search depth.

### Depth First Search (DFS)

DFS is also an important type of uniform search. DFS visits all the vertices in the graph. This type of algorithm always chooses to go deeper into the graph. After DFS visited all the reachable vertices from a particular sources vertices it chooses one of the remaining undiscovered vertices and continues the search. DFS reminds the space limitation of breath first search by always generating next a child of the deepest unexpanded nodded. The data structure stack or last in first out (LIFO) is used for DFS. One interesting

property of DFS is that, the discover and finish time of each vertex from a parenthesis structure. If we use one open parenthesis when a vertex is finished then the result is properly nested set of parenthesis.

## Concept:

**Step 1:** Traverse the root node.

**Step 2:** Traverse any neighbour of the root node.

**Step 3:** Traverse any neighbour of neighbour of the root node.

**Step 4:** This process will continue until we are getting the goal node.

## Algorithm:

**Step 1:** PUSH the starting node into the stack.

**Step 2:** If the stack is empty then stop and return failure.

**Step 3:** If the top node of the stack is the goal node, then stop and return success.

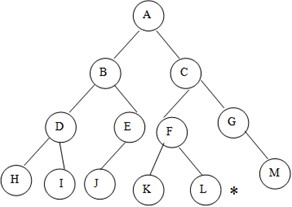
**Step 4:** Else POP the top node from the stack and process it. Find all its neighbours that are in ready state and PUSH them into the stack in any order.

**Step 5:** Go to step 3.

**Step 6:** Exit.

## Implementation:

Let us take an example for implementing the above DFS algorithm.



##### Figure Examples of DFS

Consider A as the root node and L as the goal node in the graph figure

**Step 1:** PUSH the starting node into the stack i.e.

A

**Step 2:** Now the stack is not empty and A is not our goal node. Hence move to next step.

**Step 3:** POP the top node from the stack i.e. A and find the neighbours of A i.e. B and C.

A

**Step 4:** Now C is top node of the stack. Find its neighbours i.e. F and G.

C

B

C

|  |  |  |  |
| --- | --- | --- | --- |
| B | F | G |  |

**Step 5:** Now G is the top node of the stack. Find its neighbour i.e. M

G

|  |  |  |  |
| --- | --- | --- | --- |
| B | F | M |  |

**Step 6:** Now M is the top node and find its neighbour, but there is no neighbours of M in the graph so POP it from the stack.

M

F

B

**Step 7:** Now F is the top node and its neighbours are K and L. so PUSH them on to the stack.

L F

|  |  |  |  |
| --- | --- | --- | --- |
| B | K |  |  |

**Step 8:** Now L is the top node of the stack, which is our goal node.

L

K

B

Also you can traverse the graph starting from the root A and then insert in the order C and B into the stack. Check your answer.

## Advantages:

 DFSconsumes very less memory space.

 It will reach at the goal node in a less time period than BFS if it traverses in a right path.

 It may find a solution without examining much of search because we may get the desired solution in the very first go.

## Disadvantages:

 It is possible that may states keep reoccurring.

 There is no guarantee of finding the goal node.

 Sometimes the states may also enter into infinite loops.

### Difference between BFS and DFS

**BFS**

 It uses the data structure queue.

 BFS is complete because it finds the solution if one exists.

 BFS takes more space i.e. equivalent to o (bd) where b is the maximum breath exist in a search tree and d is the maximum depth exit in a search tree.

 In case of several goals, it finds the best one.

**DFS**

 It uses the data structure stack.

 It is not complete because it may take infinite loop to reach at the goal node.

 The space complexity is O (d).

 In case of several goals, it will terminate the solution in any order.

**INFORMED SEARCH (HEURISTIC SEARCH)**

Heuristic is a technique which makes our search algorithm more efficient. Some heuristics help to guide a search process without sacrificing any claim to completeness and some sacrificing it. Heuristic is a problem specific knowledge that decreases expected search efforts. It is a technique which sometimes works but not always. Heuristic search algorithm uses information about the problem to help directing the path through the search space. These searches uses some functions that estimate the cost from the current state to the goal presuming that such function is efficient. A heuristic function is a function that maps from problem state descriptions to measure of desirability usually represented as number. The purpose of heuristic function is to guide the search process in the most profitable directions by suggesting which path to follow first when more than is available.

Generally heuristic incorporates domain knowledge to improve efficiency over blind search. In AI heuristic has a general meaning and also a more specialized technical meaning. Generally a term heuristic is used for any advice that is effective but is not guaranteed to work in every case. For example in case of

travelling sales man (TSP) problem we are using a heuristic to calculate the nearest neighbour. Heuristic is a method that provides a better guess about the correct choice to make at any junction that would be achieved by random guessing. This technique is useful in solving though problems which could not be solved in any other way. Solutions take an infinite time to compute.

Let us see some classifications of heuristic search.

### Best First Search

Best first search is an instance of graph search algorithm in which a node is selected for expansion based o evaluation function f (n). Traditionally, the node which is the lowest evaluation is selected for the explanation because the evaluation measures distance to the goal. Best first search can be implemented within general search frame work via a priority queue, a data structure that will maintain the fringe in ascending order of f values. This search algorithm serves as combination of depth first and breadth first search algorithm. Best first search algorithm is often referred greedy algorithm this is because they quickly attack the most desirable path as soon as its heuristic weight becomes the most desirable.

## Concept:

**Step 1:** Traverse the root node

**Step 2:** Traverse any neighbour of the root node, that is maintaining a least distance from the root node and insert them in ascending order into the queue.

**Step 3:** Traverse any neighbour of neighbour of the root node, that is maintaining a least distance from the root node and insert them in ascending order into the queue

**Step 4:** This process will continue until we are getting the goal node

## Algorithm:

**Step 1:** Place the starting node or root node into the queue.

**Step 2:** If the queue is empty, then stop and return failure.

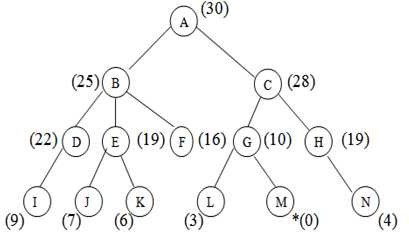
**Step 3:** If the first element of the queue is our goal node, then stop and return success.

**Step 4:** Else, remove the first element from the queue. Expand it and compute the estimated goal distance for each child. Place the children in the queue in ascending order to the goal distance.

**Step 5:** Go to step-3

**Step 6:** Exit.

## Implementation:



Let us solve an example for implementing above BFS algorithm.

##### Figure

**Step 1:**

Consider the node A as our root node. So the first element of the queue is A whish is not our goal node, so remove it from the queue and find its neighbour that are to inserted in ascending order.

A

##### Step 2:

The neighbours of A are B and C. They will be inserted into the queue in ascending order.

A

##### Step 3:

C

B

Now B is on the FRONT end of the queue. So calculate the neighbours of B that are maintaining a least distance from the roof.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| F | E | D | C |  |

B

##### Step 4:

Now the node F is on the FRONT end of the queue. But as it has no further children, so remove it from the queue and proceed further.

F

E

C

D

##### Step 5:

Now E is the FRONT end. So the children of E are J and K. Insert them into the queue in ascending order.

E

##### Step 6:

C

D

J

K

Now K is on the FRONT end and as it has no further children, so remove it and proceed further

K

##### Step7:

C

D

J

Also, J has no corresponding children. So remove it and proceed further.

J

##### Step 8:

C

D

Now D is on the FRONT end and calculates the children of D and put it into the queue.

D

I C

##### Step9:

Now I is the FRONT node and it has no children. So proceed further after removing this node from the queue.

I

C

##### Step 10:

Now C is the FRONT node .So calculate the neighbours of C that are to be inserted in ascending order into the queue.

C

H

G

##### Step 11:

Now remove G from the queue and calculate its neighbour that is to insert in ascending order into the queue.

G

L H

M

##### Step12:

Now M is the FRONT node of the queue which is our goal node. So stop here and exit.

M

H

L

## Advantage:

 It is more efficient than that of BFS and DFS.

 Time complexity of Best first search is much less than Breadth first search.

 The Best first search allows us to switch between paths by gaining the benefits of both breadth first and depth first search. Because, depth first is good because a solution can be found without computing all nodes and Breadth first search is good because it does not get trapped in dead ends.

## Disadvantages:

Sometimes, it covers more distance than our consideration.

### A\* SEARCH

A\* is a cornerstone name of many AI systems and has been used since it was developed in 1968 by Peter Hart; Nils Nilsson and Bertram Raphael. It is the combination of Dijkstra’s algorithm and Best first search. It can be used to solve many kinds of problems. A\* search finds the shortest path through a search space to goal state using heuristic function. This technique finds minimal cost solutions and is directed to a goal state called A\* search. In A\*, the \* is written for optimality purpose. The A\* algorithm also finds the lowest cost path between the start and goal state, where changing from one state to another requires some cost. A\* requires heuristic function to evaluate the cost of path that passes through the particular state. This algorithm is complete if the branching factor is finite and every action has fixed cost. A\* requires heuristic function to evaluate the cost of path that passes through the particular state. It can be defined by following formula.

f (n) = g (n) + h (n)

Where

g (n): The actual cost path from the start state to the current state. h (n): The actual cost path from the current state to goal state.

f (n): The actual cost path from the start state to the goal state.

For the implementation of A\* algorithm we will use two arrays namely OPEN and CLOSE.

##### OPEN:

An array which contains the nodes that has been generated but has not been yet examined.

##### CLOSE:

An array which contains the nodes that have been examined.

## Algorithm:

**Step 1:** Place the starting node into OPEN and find its f (n) value.

**Step 2:** Remove the node from OPEN, having smallest f (n) value. If it is a goal node then stop and return success.

**Step 3:** Else remove the node from OPEN, find all its successors.

**Step 4:** Find the f (n) value of all successors; place them into OPEN and place the removed node into CLOSE.

**Step 5:** Go to Step-2.

**Step 6:** Exit.

## Implementation:

The implementation of A\* algorithm is 8-puzzle game.

## Advantages:

 It is complete and optimal.

 It is the best one from other techniques.

 It is used to solve very complex problems.

 It is optimally efficient, i.e. there is no other optimal algorithm guaranteed to expand fewer nodes than A\*.

## Disadvantages:

 This algorithm is complete if the branching factor is finite and every action has fixed cost.

 The speed execution of A\* search is highly dependant on the accuracy of the heuristic algorithm that is used to compute h (n).

 It has complexity problems.

**Hill Climbing**

Hill climbing search algorithm is simply a loop that continuously moves in the direction of increasing value. It stops when it reaches a “peak” where no neighbour has higher value. This algorithm is considered to be one of the simplest procedures for implementing heuristic search. The hill climbing comes from that idea if you are trying to find the top of the hill and you go up direction from where ever you are. This heuristic combines the advantages of both depth first and breadth first searches into a single method.

The name hill climbing is derived from simulating the situation of a person climbing the hill. The person will try to move forward in the direction of at the top of the hill. His movement stops when it reaches at

the peak of hill and no peak has higher value of heuristic function than this. Hill climbing uses knowledge about the local terrain, providing a very useful and effective heuristic for eliminating much of the unproductive search space. It is a branch by a local evaluation function. The hill climbing is a variant of generate and test in which direction the search should proceed. At each point in the search path, a successor node that appears to reach for exploration.

## Algorithm:

**Step 1:** Evaluate the starting state. If it is a goal state then stop and return success.

**Step 2:** Else, continue with the starting state as considering it as a current state.

**Step 3:** Continue step-4 until a solution is found i.e. until there are no new states left to be applied in the current state.

##### Step 4:

* 1. Select a state that has not been yet applied to the current state and apply it to produce a new state.
  2. Procedure to evaluate a new state.
     1. If the current state is a goal state, then stop and return success.
     2. If it is better than the current state, then make it current state and proceed further.
     3. If it is not better than the current state, then continue in the loop until a solution is found.

**Step 5:** Exit.

## Advantages:

 Hill climbing technique is useful in job shop scheduling, automatic programming, circuit designing, and vehicle routing and portfolio management.

 It is also helpful to solve pure optimization problems where the objective is to find the best state according to the objective function.

 It requires much less conditions than other search techniques.

## Disadvantages:

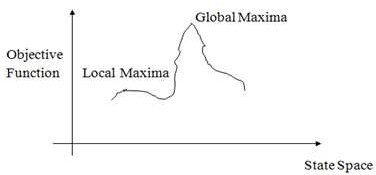
The question that remains on hill climbing search is whether this hill is the highest hill possible. Unfortunately without further extensive exploration, this question cannot be answered. This technique works but as it uses local information that’s why it can be fooled. The algorithm doesn’t maintain a search tree, so the current node data structure need only record the state and its objective function value. It assumes that local improvement will lead to global improvement.

There are some reasons by which hill climbing often gets suck which are stated below.

## Local Maxima:

A local maxima is a state that is better than each of its neighbouring states, but not better than some other states further away. Generally this state is lower than the global maximum. At this point, one cannot decide easily to move in which direction! This difficulties can be extracted by the process of backtracking

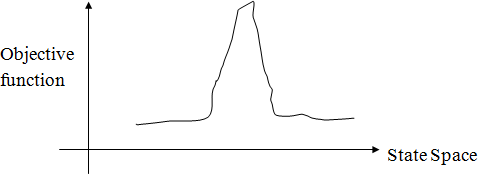
* 1. backtrack to any of one earlier node position and try to go on a different event direction. To implement this strategy, maintaining in a list of path almost taken and go back to one of them. If the path was taken that leads to a dead end, then go back to one of them.



##### Figure Local Maxima

**Ridges:**

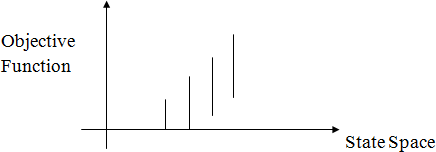
It is a special type of local maxima. It is a simply an area of search space. Ridges result in a sequence of local maxima that is very difficult to implement ridge itself has a slope which is difficult to traverse. In this type of situation apply two or more rules before doing the test. This will correspond to move in several directions at once.



##### Figure Ridges

**Plateau:**

It is a flat area of search space in which the neighbouringhave same value. So it is very difficult to calculate the best direction. So to get out of this situation, make a big jump in any direction, which will help to move in a new direction this is the best way to handle the problem like plateau.



##### Figure Plateau

**KNOWLEDGE**

Knowledge is the collection of facts, inference rules etc. which can be used for a particular purpose. Knowledge requires the use of data and information. It combines relationships, correlations, dependencies with data and information.

The basic components of knowledge are:

* + 1. A set of collected data
    2. A form of belief or hypothesis
    3. A kind of information.

Knowledge is different from data. Data is the collection of raw materials where as knowledge is the collection of some well specified inference rules and facts. Knowledge is also different from belief and hypothesis. Belief is any meaningful and coherent expression that can be represented. Belief may be true or false. A hypothesis is a justified belief that is not known to be true. A hypothesis is a belief which is backed up with some supporting evidence but it may still be false. So knowledge can be defined as true justified knowledge.

## KNOWLEDGE BASED SYSTEMS

Knowledge based systems get their power from the expert knowledge that has been coded into facts, rules, heuristics and procedures. The knowledge is stored in a knowledge base separate from the control and inferencing components. Knowledge is important and essential for knowledge based intelligent behaviour.

Input, Output Unit

Inference Control Unit

Knowledge Base

##### Figure A typical Knowledge based system

Any choice of representation will depend on the type of problem to be solved and the inference methods available. Knowledge may be vague, contradictory or incomplete. Thus, knowledge is information about objects, concepts and relationships that are assumed to exist in a particular area of interest.

## TYPE OF KNOWLEDGE

The categorisation of knowledge is very much large and interesting. They can be of following types:

### Declarative knowledge

It is the passive knowledge expressed as statements of facts about the world. It gives the simple facts and ideas about any phenomenon. It means just the representation of facts or assertions. This tells the total description about the situation. For example, the facts about an organization may be its buildings, location, no. of departments, no. of employees etc. The facts may be of two types i.e. static and dynamic. The static facts do not change with time where as the dynamic facts change with time. For example, the name and location of an organization is permanent. But some additional departments may be added.

### Procedural knowledge

Procedural knowledge is the compiled knowledge related to the performance of some task. For example the steps used to solve an algebric equation can be expressed as procedural knowledge. It also eradicates the limitations of declarative knowledge i.e. declarative knowledge tells about the organization but it cannot tell how the employees are working in that organization and how the products are developed. But procedural knowledge describes everything about the organization by using production rules and dynamic attributes.

For example, If: All the employees are very hardworking

They are very punctual They have productive ideas.

Then: Large no. of products can be produced within a very limited time period.

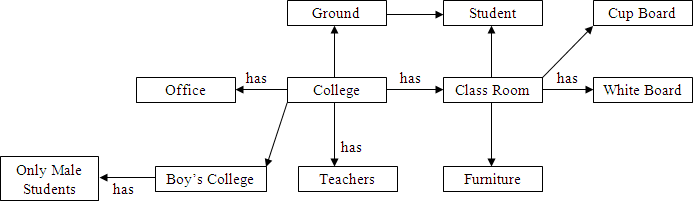
The advantages of using procedural knowledge are as follows:

1. Domain specific knowledge can be easily represented.
2. Extended logical inferences, such as default reasoning facilitated.
3. Side effects of actions may be modeled. Some disadvantages of procedural knowledge are
4. Completeness: In procedural knowledge not all cases may be represented.
5. Consistency: Not all deductions may be correct.
6. Modularity: Changes in knowledge base might have far-reaching effects.

### Inheritable knowledge

There are many situations in the world, where the object of an event inherits some properties of that particular event or any other event.

For example, consider a college. A college has certain features like classrooms, teachers, play ground, furniture, students etc. Besides these, there will be some general concepts regarding the functioning of the college, like it will have time table for each class, a fee deposit plan, examination pattern, course module etc. It can have many more deep concepts like placement of students etc. Now, if we say “A is a College”, then A will automatically inherits all the features of the college. It may be possible that X has some additional features. The inheritable knowledge is diagrammatically represented below. Here, the relationship **‘has’** indicates the silent features or attributes and **‘is a’** represents the variable or instance of that type. A inherits all the properties of college and has one additional feature of having male students. In this type of knowledge, data must be organized into a hierarchy of classes. The arrows represent the point from object to its value in the diagram. Boxed nodes represent the objects and values of attributes of objects.



##### Figure College Attribute representation

**Relational Knowledge**

Relational knowledge is made up of objects consisting of attributes and corresponding associated values. In this type of knowledge, the facts are represented as set of relations in a tabular form. The table stores or captures all the hidden attributes of objects.

For example the knowledge about doctors may be as mentioned in figure .

|  |  |  |  |
| --- | --- | --- | --- |
| **Department** | **Qualification** | **Height** | **Age** |
| Eye | P.HD | 5.0 | 35 |
| Kidney | P.HD | 5.10 | 32 |
| Surgery | P.HD | 6.3 | 28 |
| Medicine | P.HD | 6.1 | 44 |

##### Figure Knowledge about Doctor

This form of representation is the simplest and can be used in database systems. But this representation cannot store any semantic, information. For example, from this information we cannot answer the questions like “What is the name of the doctor”? or “How many doctors are in eye department”?

### Inferential Knowledge

The knowledge, which can use inference mechanism to use this knowledge is called inferential knowledge. The inheritance property is a very powerful form of inferential knowledge. The inference procedures implement the standard logic rules of inference. There are two types of inference procedures like forward inference and backward inference. Forward inference moves from start state to goal state whereas backward inference moves from goal state to start state. In this type of knowledge several symbols are generally used like  (universal quantifier), (existential quantifier),  (arrow indicator) etc.

For example: All cats have tails

 X: cat (x)  has tail (x)

##### Advantages:

1. A set of strict rules are defined which can be used to derive more facts.
2. Truths of new statements can be verified.
3. It gives guarantee about the correctness.
4. Many inference procedures available to implement standard rules of logic.

### Heuristic Knowledge

This type of knowledge is fully experimental. This knowledge requires some judgments about any performance. One can guess a good thing and also one can think bad thing. But good performances are generally taken in heuristic knowledge. For example, suppose it is asked that “Ram will score how much percentage in his final semester?” Then the answer might be 80%, 70%, 30% or 95%. The individual answers of this question based on the heuristic knowledge. The answer would be based on various factors such as past performance, his talent etc. If his previous semester percentage was 78%, then if one will say he will secure 10% in this semester then obviously he has not any knowledge about Ram.

### Tacit Knowledge

This kind of knowledge is acquired by experience. Tacit knowledge is subconsciously understood and applied, difficult to articulate and formalize. This type of knowledge is developed from direct experience and action. This knowledge is usually shared through highly interactive conversation, story telling and experience. It also includes cognitive skills such as intuition as well as technical skills such as craft and know-how. Tacit knowledge cannot be transmitted before it is converted into words, models or numbers that can be understood. Tacit knowledge can be defined in two dimensions, such as technical dimension and cognitive dimension. In technical dimension highly subjective and personal insights, intuitions and inspirations derived from long experience. The dimensions such as beliefs, ideals, principles, values and emotions fall in the category of cognitive dimension.

### Explicit Knowledge

This knowledge is formalized, coded in several natural languages (English, Italian and Spanish) or artificial languages (UML, Mathematics etc). This knowledge can be easily transmitted. It includes theoretical approaches, problem solving, manuals and database. As explicit knowledge, it was the first to be or, at least, to be archived. Tacit and explicit knowledge are not totally separate, but mutually complementary entities. Without any experience, we cannot truly understand. Explicit knowledge is playing an increasingly large role in organization and it is considered by some to be the most important factor of production in the knowledge economy. Imagine an organization without procedure manuals product literature or computer software. Also with explicit knowledge, some tacit knowledge is required to run the business in an organization. Without explicit knowledge, the organization is simply has a zero performance.

### Research Knowledge

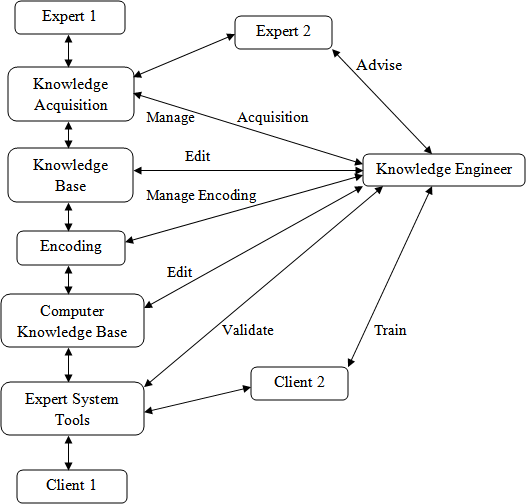
There are many standards for the generation and critical appraisal of research knowledge, but judging the quality of knowledge in this source is not without difficulty. There are disputes about the nature and content of standards in areas such as qualitative research, and the implementation of standards is

sometimes weak so that conformity with them is not necessarily a guarantee of quality. This type of knowledge is very useful for researchers to improve the research quality.

## KNOWLEDGE ACQUISITION

Knowledge acquisition is the gathering or collecting knowledge from various sources. It is the process of adding new knowledge to a knowledge base and refining or improving knowledge that was previously acquired. Acquisition is the process of expanding the capabilities of a system or improving its performance at some specified task. So it is the goal oriented creation and refinement of knowledge. Acquired knowledge may consist of facts, rules, concepts, procedures, heuristics, formulas, relationships, statistics or any other useful information. Source of these knowledges may be experts in the domain of interest, text books, technical papers, database reports, journals and the environments. The knowledge acquisition is a continuous process and is spread over entire lifetime. Example of knowledge acquisition is machine learning. It may be process of autonomous knowledge creation or refinements through the use of computer programs. The newly acquired knowledge should be integrated with existing knowledge in some meaningful way. The knowledge should be accurate, non-redundant, consistent and fairly complete. Knowledge acquisition supports the activities like entering the knowledge and maintaining knowledge base. The knowledge acquisition process also sets dynamic data structures for existing knowledge to refine the knowledge.

The role of knowledge engineer is also very important with respect to develop the refinements of knowledge. Knowledge engineers may be the professionals who elicit knowledge from experts. They integrate knowledge from various sources like creates and edits code, operates the various interactive tools, build the knowledge base etc.



##### Figure Knowledge Engineer’s Roles in Interactive Knowledge Acquisition

**Knowledge Acquisition Techniques**

Many techniques have been developed to deduce knowledge from an expert. They are termed as knowledge acquisition techniques. They are:

* 1. Diagram Based Techniques
  2. Matrix Based Techniques
  3. Hierarchy-Generation Techniques
  4. Protocol Analysis Techniques
  5. Protocol Generation Techniques
  6. Sorting Techniques

In diagram based techniques the generation and use of concept maps, event diagrams and process maps. This technique captures the features like “why, when, who, how and where”. The matrix based techniques involve the construction of grids indicating such things as problems encountered against possible

solutions. Hierarchical techniques are used to build hierarchical structures like trees. Protocol analysis technique is used to identify the type of knowledge like goals, decisions, relationships etc. The protocol generation techniques include various types of interviews like structured, semi-structured and unstructured.

The most common knowledge acquisition technique is face-to-face interview. Interview is a very important technique which must be planned carefully. The results of an interview must be verified and validated. Some common variations of an unstructured interview are talk through, teach through and read through. The knowledge engineer slowly learns about the problem. Then can build a representation of the knowledge. In unstructured interviews, seldom provides complete or well-organized descriptions of cognitive processes because the domains are generally complex. The experts usually find it very difficult to express some more important knowledge. Data acquired are often unrelated, exists at varying levels of complexity, and are difficult for the knowledge engineer to review, interpret and integrate. But on the other hand structured interviews are systematic goal oriented process. It forces an organized communication between the knowledge engineer and the expert. In structured interview, inter personal communication and analytical skills are important.

## KNOWLEDGE REPRESENTATION

Knowledge representation is probably, the most important ingredient for developing an AI. A representation is a layer between information accessible from outside world and high level thinking processes. Without knowledge representation it is impossible to identify what thinking processes are, mainly because representation itself is a substratum for a thought.

The subject of knowledge representation has been messaged for a couple of decades already. For many applications, specific domain knowledge is required. Instead of coding such knowledge into a system in a way that it can never be changed (hidden in the overall implementation), more flexible ways of representing knowledge and reasoning about it have been developed in the last 10 years.

The need of knowledge representation was felt as early as the idea to develop intelligent systems. With the hope that readers are well conversant with the fact by now, that intelligent requires possession of knowledge and that knowledge is acquired by us by various means and stored in the memory using some representation techniques. Putting in another way, knowledge representation is one of the many critical aspects, which are required for making a computer behave intelligently. Knowledge representation refers to the data structures techniques and organizing notations that are used in AI. These include semantic networks, frames, logic, production rules and conceptual graphs.

## Properties for knowledge Representation

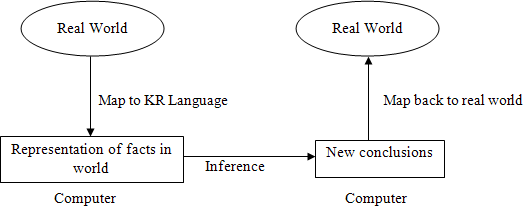
The following properties should be possessed by a knowledge representation system.

1. **Representational Adequacy:** It is the ability to represent the required knowledge.
2. **Inferential Adequacy:** It is the ability to manipulate the knowledge represented to produce new knowledge corresponding to that inferred from the original.
3. **Inferential Efficiency:** The ability to direct the inferential mechanisms into the most productive directions by storing appropriate guides.
4. **Acquisitional Efficiency:** The ability to acquire new knowledge using automatic methods wherever possible rather than reliance on human intervention.

## Syntax and semantics for Knowledge Representation

Knowledge representation languages should have precise syntax and semantics. You must know exactly what an expression means in terms of objects in the real world. Suppose we have decided that “red 1” refers to a dark red colour, “car1” is my car, car2 is another. Syntax of language will tell you which of the following is legal: red1 (car1), red1 car1, car1 (red1), red1 (car1 & car2)?

Semantics of language tell you exactly what an expression means: for example, Pred (Arg) means that the property referred to by Pred applies to the object referred to by Arg. E.g., properly “dark red” applies to my car.



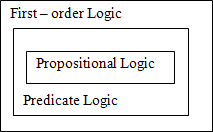
## Types of Knowledge Representation

Knowledge can be represented in different ways. The structuring of knowledge and how designers might view it, as well as the type of structures used internally are considered. Different knowledge representation techniques are

* 1. Logic
  2. Semantic Network
  3. Frame
  4. Conceptual Graphs
  5. Conceptual Dependency
  6. Script

## Logic

A logic is a formal language, with precisely defined syntax and semantics, which supports sound inference. Different logics exist, which allow you to represent different kinds of things, and which allow more or less efficient inference. The logic may be different types like propositional logic, predicate logic, temporal logic, description logic etc. But representing something in logic may not be very natural and inferences may not be efficient.



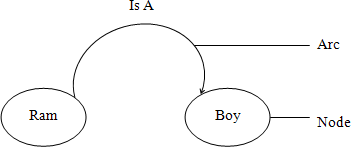
##### Figure

**Semantic Network**

A semantic network is a graphical knowledge representation technique. This knowledge representation system is primarily on network structure. The semantic networks were basically developed to model human memory. A semantic net consists of nodes connected by arcs. The arcs are defined in a variety of ways, depending upon the kind of knowledge being represented.

The main idea behind semantic net is that the meaning of a concept comes, from the ways in which it is connected to other concepts. The semantic network consists of different nodes and arcs. Each node should contain the information about objects and each arc should contain the relationship between objects. Semantic nets are used to find relationships among objects by spreading activation about from each of two nodes and seeing where the activation met this process is called intersection search.

For example: Ram is a boy.



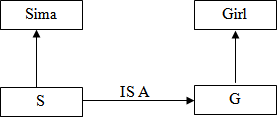
##### Figure

**Semantic network by using Instances**

The semantic network based knowledge representation mechanism is useful where an object or concept is associated with many attributes and where relationships between objects are important. Semantic nets have also been used in natural language research to represent complex sentences expressed in English. The semantic representation is useful because it provides a standard way of analyzing the meaning of sentence. It is a natural way to represent relationships that would appear as ground instances of binary predicates in predicate logic. In this case we can create one instance of each object. In instance based semantic net representations some keywords are used like: IS A, INSTANCE, AGENT, HAS-PARTS etc.

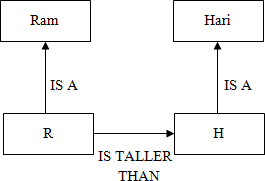
Consider the following examples:

1. Suppose we have to represent the sentence “Sima is a girl”.

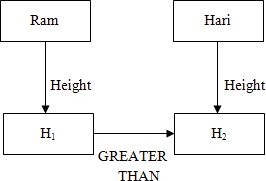


##### Figure

1. Ram is taller than Hari



It can also be represented as



##### (b)

1. “Mouse is a Rodent and Rodent is a mammal. Mouse has teeth and etas grass”. Check whether the sentence mammal has teeth is valid or not. ]

##### (c)

**Partitioned Semantic Network**

Some complex sentences are there which cannot be represented by simple semantic nets and for this we have to follow the technique partitioned semantic networks. Partitioned semantic net allow for

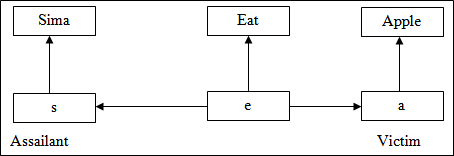
* 1. Propositions to be made without commitment to truth.
  2. Expressions to be quantified.

In partitioned semantic network, the network is broken into spaces which consist of groups of nodes and arcs and regard each space as a node.

Let us consider few examples.

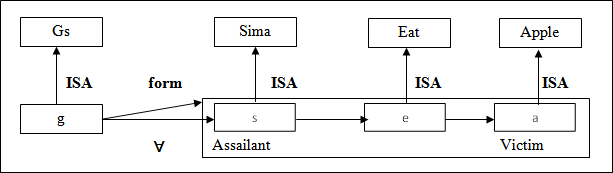
Draw the partitioned semantic network structure for the followings:

1. Sima is eating an apple.



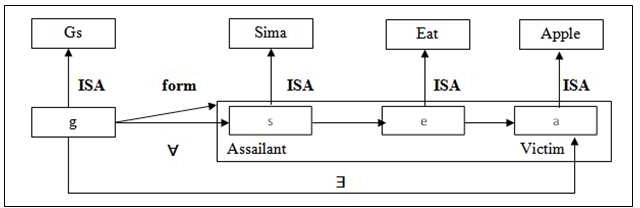
##### Figure

1. All Sima are eating an apple.



##### Figure

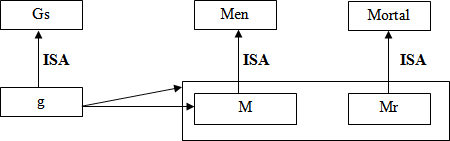
1. All Sima are eating some apple.



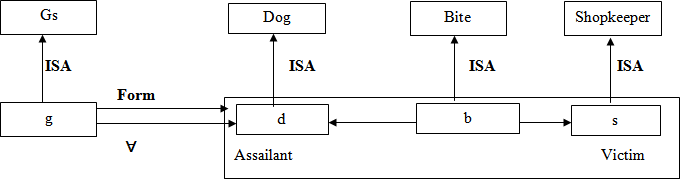
##### Figure

1. All men are mortal

##### Figure

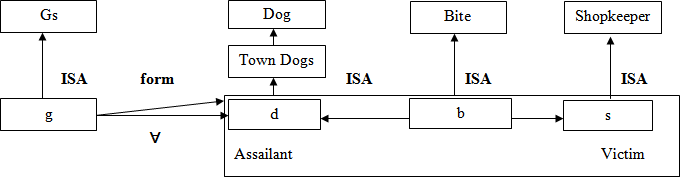


1. Every dog has bitten a shopkeeper



##### Figure

1. Every dog in town has bitten a shopkeeper.



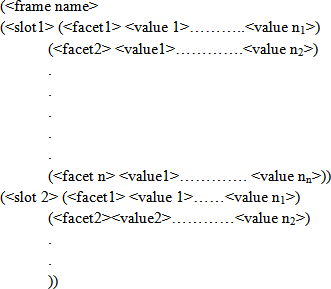
##### Figure

**NOTE:** On the above semantic network structures, the instance “IS A” is used. Also two terms like assailant and victim are used. Assailant means “by which the work is done” and that of victim refers to “on which the work is applied”. Another term namely GS, which refers to General Statement. For GS, make a node g which is an instance of Gs. Every element will have at least two attributes. Firstly, a form that states which a relation is being asserted. Secondly, one or more for all () or there exists () connections which represent universally quantifiable variables.

## FRAME

A frame is a collection of attributes and associated values that describe some entity in the world. Frames are general record like structures which consist of a collection of slots and slot values. The slots may be of any size and type. Slots typically have names and values or subfields called facets. Facets may also have names and any number of values. A frame may have any number of slots, a slot may have any number of facets, each with any number of values. A slot contains information such as attribute value pairs, default values, condition for filling a slot, pointers to other related frames and procedures that are activated when needed for different purposes. Sometimes a frame describes an entity in some absolute sense, sometimes it represents the entity from a particular point of view. A single frame taken alone is rarely useful. We build frame systems out of collection of frames that are connected to each other by virtue of the fact that the value of an attribute of one frame may be another frame. Each frame should start with an open parenthesis and closed with a closed parenthesis.

##### Syntax of a frame



Let us consider the below examples.

##### Create a frame of the person Ram who is a doctor. He is of 40. His wife name is Sita. They have two children Babu and Gita. They live in 100 kps street in the city of Delhi in India. The zip code is 756005.

(Ram

(PROFESSION (VALUE Doctor)) (AGE (VALUE 40))

(WIFE (VALUE Sita)) (CHILDREN (VALUE Bubu, Gita)) (ADDRESS

(STREET (VALUE 100 kps)) (CITY(VALUE Delhi)) (COUNTRY(VALUE India)) (ZIP (VALUE 756005))))

##### Create a frame of the person Anand who is a chemistry professor in RD Women’s College. His wife name is Sangita having two children Rupa and Shipa.

(Anand

(PROFESSION (VALUE Chemistry Professor)) (ADDRESS (VALUE RD Women’s College)) (WIFE (VALUE Sangita)) (CHILDREN(VALUE RupaShipa)))

##### Create a frame of the person Akash who has a white maruti car of LX-400 Model. It has 5 doors. Its weight is 225kg, capacity is 8, and mileage is 15 km /lit.

(Akash

(CAR (VALUE Maruti)) (COLOUR (VALUE White)) (MODEL (VALUE LX-400)) (DOOR (VALUE 5)) (WEIGHT (VALUE 225kg)) (CAPACITY (VALUE 8)) (MILAGE (VALUE 15km/lit)))

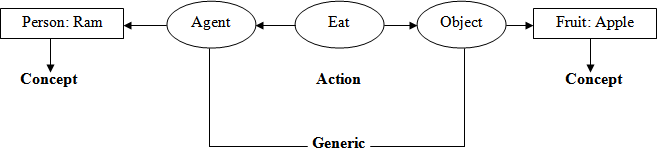
The frames can be attached with another frame and can create a network of frames. The main task of action frame is to provide the facility for procedural attachment and help in reasoning process. Reasoning using frames is done by instantiation. Instantiation process begins, when the given situation is matched with frames that are already in existence. The reasoning process tries to match the current problem state with the frame slot and assigns them values.The valuesassigned to the slots depict a particular situation and by this, the reasoning process moves towards a goal. The reasoning process can be defined as filling slot values in frames.

## Conceptual Graphs

It is a knowledge representation technique which consists of basic concepts and the relationship between them. As the name indicates, it tries to capture the concepts about the events and represents them in the form of a graph. A concept may be individual or generic. An individual concept has a type field followed by a reference field. For example person : Ram. Here person indicates type and Ram indicates reference.

An individual concept should be represented within a rectangle in graphical representation and within a square bracket in linear representation. The generic concept should be represented within an oval in graphical representation and within a parenthesis in linear representation. Conceptual graph is a basic building block for associative network. Concepts like AGENT, OBJECT, INSTRUMENT, PART are obtained from a collection of standard concepts. New concepts and relations can be defined from these basic ones. These are also basic building block for associative network. A linear conceptual graph is an elementary form of this structure. A single conceptual graph is roughly equivalent to a graphical diagram of a natural language sentence where the words are depicted as concepts and relationships.

Consider an example

“Ram is eating an apple “

##### Figure Graphical Representation



**Conceptual Dependency**

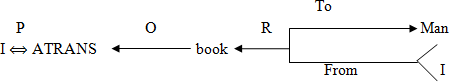
It is an another knowledge representation technique in which we can represent any kind of knowledge. It is based on the use of a limited number of primitive concepts and rules of formation to represent any natural language statement. Conceptual dependency theory is based on the use of knowledge representation methodology was primarily developed to understand and represent natural language structures. The conceptual dependency structures were originally developed by Roger C SChank in 1977.

If a computer program is to be developed that can understand wide phenomenon represented by natural languages, the knowledge representation should be powerful enough to represent these concepts. The conceptual dependency representation captures maximum concepts to provide canonical form of meaning of sentences. Generally there are four primitives from which the conceptual dependency structure can be described. They are

1. ACTS : Actions
2. PPs : Objects (Picture Producers)
3. AAs : Modifiers of Actions (Action Aiders)
4. PAs : Modifiers of PPs (Picture Aiders)
5. TS : Time of action

Conceptual dependency provides both a structure and a specific set of primitives at a particular level of granularity, out of which representation of particular pieces of information can be constructed.

For example



Where ←: Direction of dependency

Double arrow indicates two way link between actor and action. P: Past Tense

ATRANS: One of the primitive acts used by the theory O: The objective case relation

R: Recipient case Relation

In CD, representation of actions are built from a set of primitive acts.

1. **ATRANS:** Transfer of an abstract relationship (give, accept, take)
2. **PTRANS:** Transfer the physical location of an object ( Go, Come, Run, Walk)
3. **MTRANS:** Transfer the mental information (Tell)
4. **PROPEL:** Application of physical force to an object (push, pull, throw)
5. **MOVE:** Movement of a body part by its owner (kick).
6. **GRASP:** Grasping of an object by an action (clutch)
7. **INGEST:** Ingestion of an object by an animal (eat)
8. **EXPEL:** Expel from an animal body (cry)
9. **MBUILD:** Building new information out of old (decide)
10. **SPEAK:** Production of sounds (say)
11. **ATTEND:** Focusing of a sense organ towards a stimulus (Listen)

The main goal of CD representation is to capture the implicit concept of a sentence and make it explicit. In normal representation of the concepts, besides actor and object, other concepts of time, location, source and destination are also mentioned. Following conceptual tenses are used in CD representation.

1. O : Object case relationship
2. R : Recipient case relationship
3. P : Past
4. F : Future
5. Nil : Present
6. T : Transition
7. Ts : Start Transition
8. Tf : Finisher Transition
9. K : Continuing
10. ? : Interrogative
11. / : Negative
12. C : Conditional

Also there are several rules in conceptual dependency

**Rule 1:** PP ACT

It describes the relationship between an actor and an event, he/she causes.

E.g. Ram ran

Ram PTRANS

Where P: Past Tense

**Rule 2:** PP  PA

It describes the relationship between a PP and PA where the PA indicates one characteristics of PP. E.g. Ram is tall

Ram Tall or Ram Height (> Average)



**Rule 3:** PP PP

It describes the relationship between two PPs where one PP is defined by other.

E.g. Ram is a doctor Ram  Doctor

**Rule 4:** PP or PA

PA PP

It describes the relationship between the PP and PA, where PA indicates one attributes of PP.

E.g. A nice boy is a doctor Boy Doctor

Nice

##### Rule 5: PP

PP

It describes the relationship between 3 PP’s where one PP is the owner of another PP.

E.g. Ram’s Cat Cat

Ram

**Rule 6:** Act PP Where O: Object

It describes the relationship between the PP and ACT. Where PP indicates the object of that action. E.g. Ram is eating an apple.

Ram

**Rule 7:** ACT (R: Recipient)

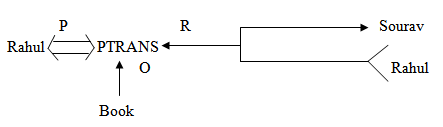
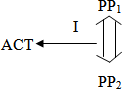
INGEST O

Apple



Here one PP describes the recipient and another PP describes the donner

E.g. Rahul gave a book to sourav.

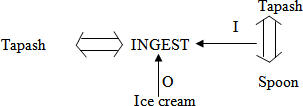


**Rule 8:**

(I: Instrument used in the action)

Here PP1 indicates the agent and PP2 indicates the object that is used in the action.

E.g. Tapash ate the ice cream with the spoon.



##### Rule 9:

Here D indicates destination, PP1 indicates destination and PP2 indicates the source.

E.g. the bucket is filled with milk.



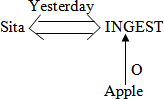
xindicates the average milk and the source i.e. bucket is dry which is hidden.

##### Rule 10:

(T: Time)

It describes the relationship between a conceptualization and the time at which the event is described occurs.

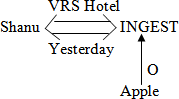
E.g. Sita ate the apple yesterday.



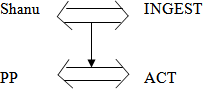
##### Rule 11:

It describes the relationship between a conceptualization and the place at which it is occurred.

E.g. Shanu ate the apple at VRS hotel yesterday

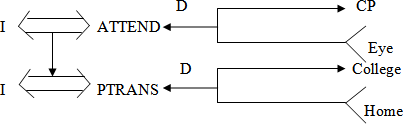


##### Rule 12:



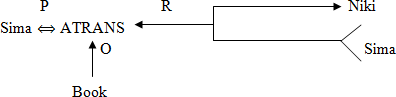
It describes the relationship between one conceptualization with another.

E.g. while I was going to college, I saw a snake



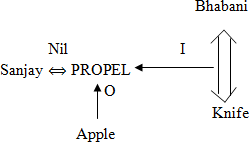
(Where CP: Conscious Processor i.e. the combination of all sense organs like eye, ear, nose etc.)

By using the above rules we can represent any sentence. Let us visualize few examples on conceptual dependency.

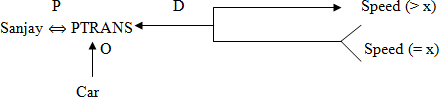
* 1. Sima gave a book to Niki

Where O: Object, P: Past Tense, R: Recipient, Sima: PP, Book: PP, Niki: PP, ATRANS: give

* 1. Bhabani cuts an apple with a knife

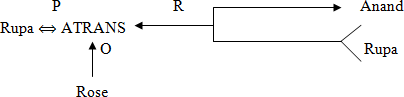


* 1. Sanjay drove the car fast

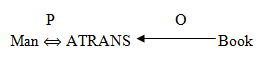
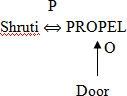


The rose was given by Rupa to Anand

4)



5) Shruti pushed the door.



6) The man took a book

Here man is the doctor and book is the object of the action took.

1. My grandfather told me a story



1. Ira gave the man a dictionary



##### SCRIPT

It is an another knowledge representation technique. Scripts are frame like structures used to represent commonly occurring experiences such as going to restaurant, visiting a doctor. A script is a structure that describes a stereotyped sequence of events in a particular context. A script consist of a set of slots. Associated with each slot may be some information about what kinds of values it may contain as well as a default value to be used if no other information is available. Scripts are useful because in the real world, there are no patterns to the occurrence of events. These patterns arise because of clausal relationships between events. The events described in a script form a giant casual chain. The beginning of the chain is the set of entry conditions which enable the first events of the script to occur. The end of the chain is the set of results which may enable later events to occur. The headers of a script can all serve as indicators that the script should be activated.

Once a script has been activated, there are a variety of ways in which it can be useful in interpreting a particular situation. A script has the ability to predict events that has not explicitly been observed. An important use of scripts is to provide a way of building a single coherent interpretation from a collection of observation. Scripts are less general structures than are frames and so are not suitable for representing all kinds of knowledge. Scripts are very useful for representing the specific kinds of knowledge for which they were designed.

A script has various components like:

1. **Entry condition**: It must be true before the events described in the script can occur. E.g. in a restaurant script the entry condition must be the customer should be hungry and the customer has money.
2. **Tracks:** It specifies particular position of the script e.g. In a supermarket script the tracks may be cloth gallery, cosmetics gallery etc.
3. **Result:** It must be satisfied or true after the events described in the script have occurred.

e.g. In a restaurant script the result must be true if the customer is pleased.

The customer has less money.

1. **Probs:** It describes the inactive or dead participants in the script e.g. In a supermarket script, the probes may be clothes, sticks, doors, tables, bills etc.
2. **Roles:** It specifies the various stages of the script. E.g. In a restaurant script the scenes may be entering, ordering etc.

Now let us look on a movie script description according to the above component.

* 1. Script name : Movie
  2. Track : CINEMA HALL
  3. Roles : Customer(c), Ticket seller(TS), Ticket Checker(TC), Snacks Sellers (SS)
  4. Probes : Ticket, snacks, chair, money, Ticket, chart
  5. Entry condition : The customer has money

The customer has interest to watch movie.

##### Scenes:

1. **SCENE-1 (Entering into the cinema hall)**

C PTRANS C into the cinema hall

C ATTEND eyes towards the ticket counter C PTRANS C towards the ticket counters C ATTEND eyes to the ticket chart

C MBUILD to take which class ticket C MTRANS TS for ticket

C ATRANS money to TS TS ATRANS ticket to C

##### SCENE-2 (Entering into the main ticket check gate)

C PTRANS C into the queue of the gate C ATRANS ticket to TC

TC ATTEND eyes onto the ticket

TC MBUILD to give permission to C for entering into the hall TC ATRANS ticket to C

C PTRANS C into the picture hall.

##### SCENE-3 (Entering into the picture hall)

C ATTEND eyes into the chair TC SPEAK where to sit

C PTRANS C towards the sitting position C ATTEND eyes onto the screen

##### SCENE-4 (Ordering snacks)

C MTRANS SS for snacks SS ATRANS snacks to C

C ATRANS money to SS

C INGEST snacks

##### SCENE-5 (Exit)

C ATTEND eyes onto the screen till the end of picture C MBUILD when to go out of the hall

C PTRANS C out of the hall

##### Result:

The customer is happy

The customer has less money

##### Example 2: Write a script of visiting a doctor in a hospital

1. SCRIPT\_NAME : Visiting a doctor
2. TRACKS : Ent specialist
3. ROLES : Attendant (A), Nurse(N), Chemist (C),

Gatekeeper(G), Counter clerk(CC), Receptionist(R), Patient(P), Ent specialist Doctor (D), Medicine

Seller (M).

1. PROBES : Money, Prescription, Medicine, Sitting chair,

Doctor’s table, Thermometer, Stetho scope, writing pad, pen, torch, stature.

1. ENTRY CONDITION: The patient need consultation.

Doctor’s visiting time on.

##### SCENES:

* 1. **SCENE-1 (Entering into the hospital)**

P PTRANS P into hospital

P ATTEND eyes towards ENT department P PTRANS P into ENT department

P PTRANS P towards the sitting chair

##### SCENE-2 (Entering into the Doctor’s Room)

P PTRANS P into doctor’s room P MTRANS P about the diseases P SPEAK D about the disease

D MTRANS P for blood test, urine test D ATRANS prescription to P

P PTRANS prescription to P.

P PTRANS P for blood and urine test

##### SCENE-3 (Entering into the Test Lab)

P PTRANS P into the test room

P ATRANS blood sample at collection room P ATRANS urine sample at collection room P ATRANS the examination reports

##### SCENE-4 (Entering to the Doctor’s room with Test reports)

P ATRANS the report to D

D ATTEND eyes into the report D MBUILD to give the medicines

D SPEAK details about the medicine to P

P ATRANS doctor’s fee

P PTRANS from doctor’s room

##### SCENE-5 (Entering towards medicine shop)

P PTRANS P towards medicine counter P ATRANS Prescription to M

M ATTEND eyes into the prescription M MBUILD which medicine to give M ATRANS medicines to P

P ATRANS money to M

P PTRANS P from the medicine shop

##### RESULT:

The patient has less money

Patient has prescription and medicine.

## Advantages And Disadvantages Of Different Knowledge Representation

|  |  |  |  |
| --- | --- | --- | --- |
| Sl.  No. | Scheme | Advantages | Disadvantages |
| 1 | Production rules | * Simple syntax * Easy to understand * Simple interpreter * Highly Modular * Easy to add or modify | * Hard to follow Hierarchies * Inefficient for large systems * Poor at representing structured descriptive knowledge. |
| 2 | Semantic | * Easy to follow hierarchy * Easy to trace associations * Flexible | * Meaning attached to nodes might be ambiguous * Exception handling is difficult * Difficult to program |
| 3 | Frame | * Expressive Power * Easy to set up slots for new properties and relations * Easy to create specialized | * Difficult to program * Difficult for inference * Lack of inexpensive software |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | procedures |  |
| 4 | Script | * Ability to predict events * A single coherent interpretation may be build up from a collection of   observations | * Less general than frames * May not be suitable to represent all kinds of knowledge |
| 5 | Formal Logic | * Facts asserted independently of use * Assurance that only valid consequence are asserted * Completeness | * Separation of representation and processing * Inefficient with large data sets * Very slow with large knowledge bases |

### HUMAN ASSOCIATIVE MEMORY (HAM)

This model was developed by John Anderson and Gordon Bower (1973). This memory is organized as a network of propositional binary trees. When an informant asserts a statement to HAM, the system parses the sentence and builds a binary tree representation. As HAM is informed of new sentences, they are parsed and formed into new tree like structures with existing ones. When HAM is posed with a query it is formed into a tree structure called a probe. This structure is then matched against memory structures for the best match. The structure with the closest match is used to formulate an answer to the query. Matching is accomplished by first locating the leaf nodes in memory that match leaf nodes in the probe. The corresponding links are then checked to see if they have the same labels and in the same order. The search process is constrained by searching only node groups that have the same relation links. Access to nodes in HAM is accomplished through word indexing in LISP.

In HAM, nodes in the tree are assigned with unique numbers, while links are labeled with some functions. They are given below:

C : Context for free fact E : Set membership

F : a fact

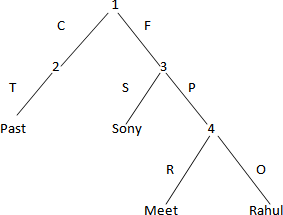
L : a location O : An object P : Predicate

R : Relation S : Subject

T : Time (Present, past, future)

On the basis of above function, we can represent various sentences in HAM. Let us look some example by using the above functions.

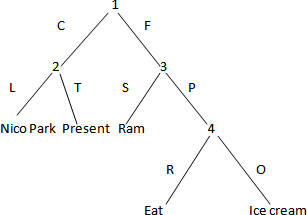
1. Sony met Rahul



##### Figure

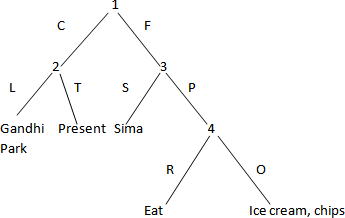
On the above Ham Structure the time is the past (met), as Sonly did the work so sonly is the subject and Rahul will be the object and the relation is met.

1. Ram is eating an apple at Nico Park.



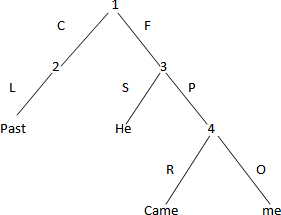
##### Figure

1. Sima is eating ice cream as well as chips at Gandhi Park.



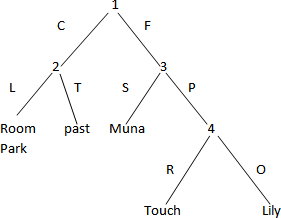
##### Figure

1. He came to me.



##### Figure

1. In a room Muna touched Lily.



##### Figure

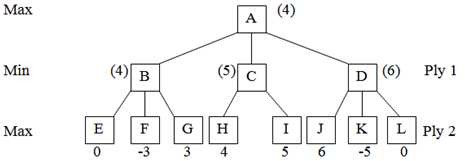
**Module 2**

**MIN-MAX Search**

Games have always been an important application area for heuristic algorithms. In playing games whose state space may be exhaustively delineated, the primary difficulty is in accounting for the actions of the opponent. This can be handled easily by assuming that the opponent uses the same knowledge of the state space as us and applies that knowledge in a consistent effort to win the game. Minmax implements game search under referred to as MIN and MAX.

The min max search procedure is a depth first, depth limited search procedure. The idea is to start at the current position and use the plausible move generator to generate the set of possible successor positions. To decide one move, it explores the possibilities of winning by looking ahead to more than one step. This is called a ply. Thus in a two ply search, to decide the current move, game tree would be explored two levels farther.

Consider the below example



##### Figure Tree showing two ply search

In this tree, node A represents current state of any game and nodes B, C and D represent three possible valid moves from state A. similarly E, F, G represents possible moves from B, H, I from C and J, K, L, from D. to decide which move to be taken from A, the different possibilities are explored to two next steps. 0, -3, 3, 4, 5, 6, -5, 0 represent the utility values of respective move. They indicate goodness of a move. The utility value is back propagated to ancestor node, according to situation whether it is max ply or min ply. As it is a two player game, the utility value is alternatively maximized and minimized. Here as

the second player’s move is maximizing, so maximum value of all children of one node will be back propagated to node. Thus, the nodes B, C, D, get the values 4, 5, 6 respectively. Again as ply 1 is minimizing, so the minimum value out of these i.e. 4 is propagated to A. then from A move will be taken to B.

MIN MAX procedure is straightforward recursive procedure that relies on two auxiliary procedures that are specific to the game being played.

* 1. MOVEGEN (position, player): the move generator which returns a list of nodes representing the moves that can be made by player in position. We may have 2 players namely PLAYER-TWO in a chess problem.
  2. STATIC (position, player): the static evaluation function, which returns a number representing the goodness of position from the standpoint of player.

We assume that MIN MAX returns a structure containing both results and that we have two functions, VALUE and PATH that extract the separate components. A function LAST PLY is taken which is assumed to evaluate all of the factors and to return TRUE if the search should be stopped at the current level and FALSE otherwise.

MIN MAX procedure takes three parameters like a board position, a current depth of the search and the players to move. So the initial call to compute the best move from the position CURRENT should be

MIN MAX (CURRENT, 0, PLAYER-ONE)

(If player is to move)

Or

MIN MAX (CURRENT, 0, PLAYER-TWO)

(If player two is to move) Let us follow the algorithm of MIN MAX

**Algorithm: MINMAX (position, depth, player)**

* + 1. If LAST PLY (position, depth)

Then RETURN VALUE = STATIC (position, player) PATH = nil.

* + 1. Else, generate one more ply of the tree by calling the function MOVE\_GEN (position, player) and set SUCCESORS to the list it returns.
    2. If SUCESSORS is empty,

THEN no moves to be made

RETURN the same structure that would have been returned if LAST\_PLY had returned TRUE.

* + 1. If SUCCESORS is not empty,

THEN examine each element in turn and keep track of the best one.

* + 1. After examining all the nodes,

RETURN VALUE = BEST- SCORE

PATH = BEST- PATH

When the initial call to MIN MAX returns, the best move from CURRENT is the first element in the PATH.

**EXPERT SYSTEM:**

An expert system may be viewed as a computer simulation of a human expert. It can also be defined as a computer program that simulates the judgment and behaviour of a human or an organization that has expert knowledge and experience in a particular field. Typically such a system contains a knowledge base containing accumulated experience and a set of rules for applying the knowledge base to each particular situation that is described to the program. Expert

systems also use human knowledge to solve problems that normally would require human intelligence. These expert systems represent the expertise knowledge as data or rules within the computer. These rules and data can be called upon when needed to solve problems. Books and manual guides have a tremendous amount of knowledge but a human has to read and interpret the knowledge for it to be used.

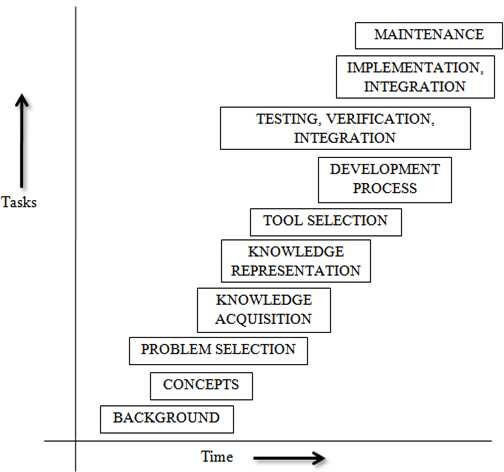
A system that uses human knowledge captured in a computer to solve problems that ordinarily require human expertise. A computer program designed to model the problem solving ability of a human expert. Expert systems make extensive use of specialized knowledge to solve problems at the level of a human expert. An expert is a person who has expertise in a certain area i.e. the expert has knowledge or special skills that are not known or available to most people. An expert can solve problems that most people cannot solve them much more efficiently. Thus expert system technology may include special expert system languages, programs and hardware designed to aid in the development and execution of expert systems. The knowledge in expert systems may be either expertise or knowledge that is generally available from books, magazines and knowledgeable persons.

**DIFFERENCE BETWEEN EXPERT SYSTEM AND CONVENTIONAL SYSTEM**

The principle distinction between expert systems and traditional problem solving programs is the way in which the problem related expertise is coded. In conventional applications, problem expertise is encoded in both program and data structures. In the expert system approach all of the problem related expertise is encoded in data structures only, none is in programs. Generally in expert systems, the use of knowledge is vital. But in conventional system data is used more efficiently than knowledge. Conventional systems are not capable of explaining a particular conclusion for a problem. These systems try to solve in a straight forward manner. But expert systems are capable of explaining how a particular conclusion is reached and why requested information is needed during a process. However, the problems are solved more efficiently than a conventional system by an expert system. Generally in an expert system, it uses the symbolic representations for knowledge i.e. the rules, different forms of networks, frames, scripts etc. and performs their inference through symbolic computations. But conventional systems are unable to express these terms. They just simplify the problems in a straight forward manner and are incapable to express the “how, why” questions. Also the problem solving tools those are present in expert system are purely absent in conventional systems. The various types of problems are always solved by the experts in an expert system. So the solution of the problem is more accurate than a conventional system.

**THE DEVELOPMENT PROCESS OF AN EXPERT SYSTEM**

By the definition, an expert system is a computer program that simulates the thought process of a human expert to solve complex decision problems in a specific domain. The expert system’s knowledge is obtained from expert sources which are coded into most suitable form. The process of building an expert system is called knowledge engineering and is done by a knowledge engineer. The knowledge engineer is a human with a background in computer science and AI and he knows how to build expert systems. A knowledge engineer also decides how to represent the knowledge in an expert system and helps the programmers to write the code. Knowledge engineering is the acquisition of knowledge from a human expert or any other source. The different stages in the development of an expert system are illustrated in figure.



##### Figure Hierarchy of expert system development process

Some latest developments in the expert system area are as follows:

1. Availability of many tools that are designed to expedite the construction of expert system at a reduced cost.
2. Increased use of expert systems in many tasks ranging from help desks to complex military and space shuttle applications.
3. Use of multiple knowledge bases.
4. Improvements in knowledge acquisition.
5. Use of the internet to disseminate software and expertise.
6. Increased use of object oriented programming approach in knowledge representation.
7. The multiple use of heuristic knowledge in several applications.
8. Enables the user to think about hypothetical reasoning.

## CHARACTERISTICS OF AN EXPERT SYSTEM

The growth of expert system is expected to continue for several years. With the continuing growth, many new and exciting applications will emerge. An expert system operates as an interactive system that responds to questions, asks for clarification, makes recommendations and generally aids the decision making process. Expert system provides expert advice and guidance in a wide variety of activities from computer diagnosis to delicate medical surgery.

An expert system is usually designed to have the following general characteristics.

1. **High level Performance:** The system must be capable of responding at a level of competency equal to or better than an expert system in the field. The quality of the advice given by the system should be in a high level integrity and for which the performance ratio should be also very high.
2. **Domain Specificity:** Expert systems are typically very domain specific. For ex., a diagnostic expert system for troubleshooting computers must actually perform all the necessary data manipulation as a human expert would. The developer of such a system must limit his or her scope of the system to just what is needed to solve the target problem. Special tools or programming languages are often needed to accomplish the specific objectives of the system.
3. **Good Reliability:** The expert system must be as reliable as a human expert.
4. **Understandable:** The system should be understandable i.e. be able to explain the steps of reasoning while executing. The expert system should have an explanation capability similar to the reasoning ability of human experts.
5. **Adequate Response time:** The system should be designed in such a way that it is able to perform within a small amount of time, comparable to or better than the time taken by a human expert to reach at a decision point. An expert system that takes a year to reach a decision compared to a human expert’s time of one hour would not be useful.
6. **Use symbolic representations:** Expert system use symbolic representations for knowledge (rules, networks or frames) and perform their inference through symbolic computations that closely resemble manipulations of natural language.
7. **Linked with Metaknowledge:** Expert systems often reason with metaknowledge i.e. they reason with knowledge about themselves and their own knowledge limits and capabilities. The use of metaknowledge is quite interactive and simple for various data representations.
8. **Expertise knowledge:** Real experts not only produce good solutions but also find them quickly. So, an expert system must be skillful in applying its knowledge to produce solutions both efficiently and effectively by using the intelligence human experts.
9. **Justified Reasoning:** This allows the users to ask the expert system to justify the solution or advice provided by it. Normally, expert systems justify their answers or advice by explaining their reasoning. If a system is a rule based system, it provides to the user all the rules and facts it has used to achieve its answer.
10. **Explaining capability:** Expert systems are capable of explaining how a particular conclusion was reached and why requested information is needed during a consultation. This is very important as it gives the user a chance to access and understand the system’s reasoning ability, thereby improving the user’s confidence in the system.
11. **Special Programming Languages:** Expert systems are typically written in special programming languages. The use of languages like LISP and PROLOG in the development of an expert system simplifies the coding process. The major advantage of these languages, as compared to conventional programming languages is the simplicity of the addition, elimination or substitution of new rules and memory management capabilities. Some of the distinguishing characteristics of programming languages needed for expert system work are as follows:
    1. Efficient mix of integer and real variables.
    2. Good memory management procedures.
    3. Extensive data manipulation routines.
    4. Incremental compilation.
    5. Tagged memory architecture.
    6. Efficient search procedures.
    7. Optimization of the systems environment.

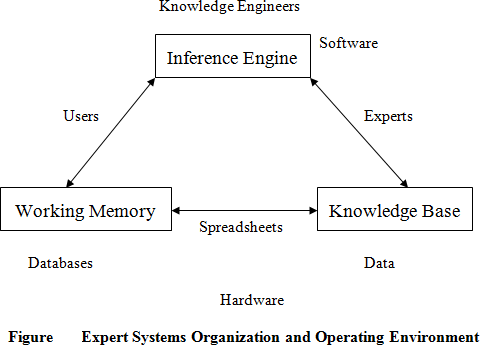
## STRUCTURE OF AN EXPERT SYSTEM

The structure of expert systems reflect the knowledge engineers understanding of the methods of representing knowledge and of how to perform intelligent decision making tasks with the support of a computer based system. Complex decisions involve intricate combination of factual and heuristic knowledge. In order for the computer to be able to retrieve and effectively use heuristic knowledge, the knowledge must be organized in an easily accessible format that distinguishes among data, knowledge and control structures. For this reason expert systems are organized in three distinct levels like:

1. **Knowledge Base:** It consists of problem solving rules, procedures and intrinsic data relevant to the problem domain. The knowledge base constitutes the problem solving rules, facts or intuition that a

human expert might use in solving problems in a given problem domain. The knowledge base is usually stored in terms of if-then rules. The working memory represents relevant data for the current problem being solved.

1. **Working Memory:** It refers to task specific data for the problem under consideration. This is the dynamic module of the system. It consists of an essential component called database. In general, the workspace contains a set called rule base, i.e. it contains a set of rules that to be used by a system at a given moment.
2. **Inference Engine:** This is a generic control mechanism that applies the axiomatic knowledge in the knowledge base to the task specific data to arrive at some solution or conclusion. Inference in production systems is accomplished by a process of chaining through the rules recursively, either in a forward or in a backward direction until a conclusion is reached.

These three pieces may very well come from different sources. The inference engine, such as VP-Expert, may come from a commercial vendor. The knowledge base may be a specific diagnostic knowledge base compiled by a consulting firm, and the problem data may be supplied by the end user. A knowledge base is the nucleus of the expert system structure. A knowledge base is created by knowledge engineers, who translate the knowledge of real human experts into rules and strategies. These rules and strategies can change depending on the prevailing problem scenario. The knowledge base provides the expert system with the capability to recommend directions for user inquiry. The system also instigates further investigation into areas that may be important to a certain line of reasoning but not apparent to the user. The general structure of an expert system is given in figure .

The

modularity of an expert system is an important distinguishing characteristics compared to a conventional computer program. Modularity is affected in an expert system by the use of three distinct components as shown in fig 6.2. A good expert system is expected to grow as it learns from user feedback. Feedback is incorporated into the knowledge base as appropriate to make the expert system smarter. The dynamism of the application environment for expert systems is based on the individual dynamism of the components. This can be classified into three categories as follows.

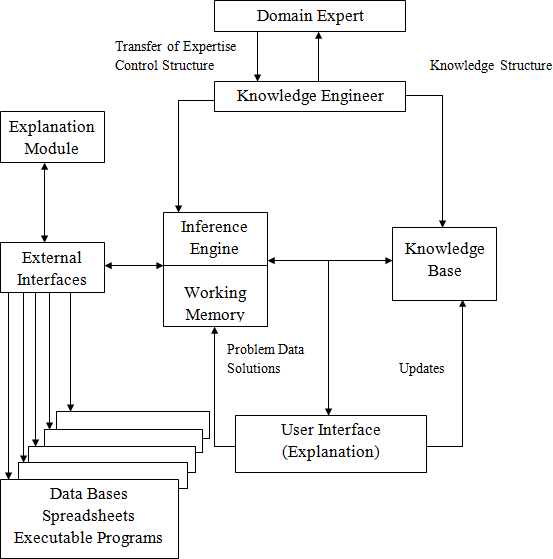
* 1. **Most dynamic:** The most dynamic part of an expert system is always the working memory. The content of the working memory, sometimes called the data structure, changes with each problem situation. Consequently, it is the most dynamic component of an expert system assuming, of course that it is kept current.
  2. **Moderately dynamic:** This part in the expert system is the knowledge base. The knowledge base need not change unless a new piece of information arises that indicates a change in the problem solution procedure. Changes in the knowledge base should be carefully evaluated before being implemented. In effect, changes should not be based on just one consultation experience.
  3. **Least dynamic:** The least dynamic part is the inference engine. As the control and coding structure of an inference engine is very strict, so changes are made only if absolutely necessary to correct a bug or enhance the inferential process. Commercial inference engines, in particular, change only at the discretion of the developer. Since frequent updates can be disruptive and costly to clients, most commercial software developers try to minimize the frequency of updates.

## RULE BASED ARCHITECTURE OF AN EXPERT SYSTEM

The most common form of architecture used in expert and other types of knowledge based systems is the production system or it is called rule based systems. This type of system uses knowledge encoded in the form of production rules i.e. if-then rules. The rule has a conditional part on the left hand side and a conclusion or action part on the right hand side. For example if: condition1 and condition2 and condition3

Then: Take action4

Each rule represents a small chunk of knowledge to the given domain of expertise. When the known facts support the conditions in the rule’s left side, the conclusion or action part of the rule is then accepted as known. The rule based architecture of an expert system consists of the domain expert, knowledge engineer, inference engine, working memory, knowledge base, external interfaces, user interface, explanation module, database spreadsheets executable programs s mentioned in figure .



##### Integration of Expert systems Components

The components of the rule based architecture are as follows.

1. **User Interface:** It is the mechanism by which the user and the expert system communicate with each other i.e. the use interacts with the system through a user interface. It acts as a bridge between user and expert system. This module accepts the user queries and submits those to the expert system. The user normally consults the expert system for following reasons.
   1. To get answer of his/her queries.
   2. To get explanation about the solution for psychological satisfaction.

The user interface module is designed in such a way that at user level it accepts the query in a language understandable by expert system. To make the expert system user friendly, the user interface interacts with the user in natural language. The user interface provides as much facilities as possible such as menus, graphical interfaces etc. to make the dialog user friendly and more attractive.

1. **Explanation Module:** The explanation module explains the reasoning of the system to a user. It provides the user with an explanation of the reasoning process when requested. The credibility of

expert system will be established only when it is able to explain “how and why” a particular conclusion is drawn. This explanation increases the belief of user in the expert system.

* 1. **Explanation(How):** To respond to a how query, the explanation module traces the chain of rules fired during a consolation with the user. This explanation mode can be activated once the process is over. It explains how a fact was deduced by the system and similarly how a rule was/wasn’t used. The simplest way to specify this is to explain the rule which allows the deduction. For e.g. If the system (S) will give information about the parent-child relationship to the user (U) then the followings can be possible.

S: My diagnosis is “A is the father of B” U: How?

S: The result was obtained by the addition of following facts and rules. Fact no 11: A is the parent of Hari.

Fact no 15: A is a male.

Fact no 110: X is father of Y:

X is parent of Y, X is male.

So A is the father of B.

* 1. **Explanation (Why)?** To respond to a why query, the explanation module must be able to explain why certain information is needed by the inference engine to complete a step in the reasoning process. This mode of explanation is dynamic and is used when the system functions in backward chaining. If the user doesn’t understand the reason behind the question. The system is capable of explaining why the question was asked.

For example S: Is the following true?

A is the father of B. U: Why?

S: I need the fact:

A is the father of B to establish the following fact “B is the son of A”.

By using the rule no. 4: A is the father of B:

B is the son of A.

1. **Working Memory:** It is a global database of facts used by the rules.
2. **Knowledge Engineering:** The primary people involved in building an expert system are the knowledge engineer, the domain expert and the end user. Once the knowledge engineer has obtained a general overview of the problem domain and gone through several problem solving sessions with

the domain expert, he/she is ready to begin actually designing the system, selecting a way to represent the knowledge, determining the search strategy (backward or forward) and designing the user interface. After making complete designs, the knowledge engineer builds a prototype. The prototype should be able to solve problems in a small area of the domain. Once the prototype has been implemented, the knowledge engineer and domain expert test and refine its knowledge by giving it problems to solve and correcting its disadvantages.

1. **Knowledge Base:** In rule based architecture of an expert system, the knowledge base is the set of production rules. The expertise concerning the problem area is represented by productions. In rule based architecture, the condition actions pairs are represented as rules, with the premises of the rules (if part) corresponding to the condition and the conclusion (then part) corresponding to the action. Case-specific data are kept in the working memory. The core part of an expert system is the knowledge base and for this reason an expert system is also called a knowledge based system. Expert system knowledge is usually structured in the form of a tree that consists of a root frame and a number of sub frames. A simple knowledge base can have only one frame, i.e. the root frame whereas a large and complex knowledge base may be structured on the basis of multiple frames.

**Inference Engine:** The inference engine accepts user input queries and responses to questions through the I/O interface. It uses the dynamic information together with the static knowledge stored in the knowledge base. The knowledge in the knowledge base is used to derive conclusions about the current case as presented by the user’s input. Inference engine is the module which finds an answer from the knowledge base. It applies the knowledge to find the solution of the problem. In general, inference engine makes inferences by deciding which rules are satisfied by facts, decides the priorities of the satisfied rules and executes the rule with the highest priority. Generally inferring process is carried out recursively in 3 stages like match, select and execute. During the match stage, the contents of working memory are compared to facts and rules contained in the knowledge base. When proper and consistent matches are found, the corresponding rules are placed in a conflict set.

## APPLICATIONS OF EXPERT SYTEM

There are several major application areas of expert system such as agriculture, education, environment, law manufacturing, medicine power system etc. Expert system is used to develop a large number of new products as well as new configurations of established products. When established products are modified to include an expert system as a component or when an established product item is replaced with an expert system, the expert system supported entity is called intelligent. Expert systems are designed and created to facilitate tasks in the fields of accounting, medicine, process control, financial service, production, education etc. The foundation of a successful expert system depends on a series of technical procedures and development that may be designed by certain related experts.

### Expert Systems are for everyone

Everyone can find an application potential in the field of expert systems. Contrary to the belief that expert systems may pose a threat to job security, expert systems can actually help to create opportunities for new job areas. No matter which is of business one is engages in, expert systems can fulfill the need for higher productivity and reliability of decisions. Some job opportunities offered by the expert system are listed below:

 Basic Research

 Applied Research

 Knowledge Engineering

 The development of Inference engine

 Training

 Sales and marketing

### Expert System in Education

In the field of education, many of the expert system’s application are embedded inside the Intelligent Tutoring System (ITS) by using techniques from adaptive hypertext and hypermedia. Most of the system usually will assist student in their learning by using adaptation techniques to personalize with the environment prior knowledge of student and student’s ability to learn. Expert system in education has expanded very consistently from micro computer to web based and agent based technology. Web based expert system can provide an excellent alternative to private tutoring at any time from any place where internet is provided. Agent based expert system will help users by finding materials from the web based on the user’s profile. Expert system also had tremendous changes in the applying of methods and techniques. Expert system are beneficial as a teaching tools because it has equipped with the unique features which allow users to ask question on how, why and what format. When it is used in the class environment, surely it will give many benefit to student as it prepare the answer without referring to the teacher. Beside that, expert system is able to give reasons towards the given answer. Expert system had been used in several fields of study including computer animation, computer science and engineering, language teaching business study etc.

### Expert system in Agriculture

The expert system for agriculture is same as like other fields. Here also the expert system uses the rule based structure and the knowledge of a human expert is captured in the form of IF-THEN rules and facts which are used to solve problems by answering questions typed at a keyboard attached to a computer. For example, in pest control, the need to spray, selection of a chemical to spray, mixing and application etc. The early, state of developing the expert systems are in the 1960’s and 1970’s were typically written on a mainframe computer in the programming language based on LISP. Some examples of these expert

systems are MACSYMA developed at the Massachusetts Institute of Technology (MIT) for assisting individuals in solving complex mathematical problems. Other examples may be MYCIN, DENDRAL, and CALEX etc. The rises of the agricultural expert system are to help the farmers to do single point decisions, which to have a well planning for before start to do anything on their land. It is used to design an irrigation system for their plantation use. Also some of the other functions of agricultural expert system are:

 To predict the extreme events such as thunderstorms and frost.

 To select the most suitable crop variety.

 Diagnosis of liver stock disorder and many more.

### Expert System for a particular decision problem

The expert system can be used as a stand alone advisory system for the specific knowledge domain. It also can provide decision support for a high level human expert. The main purposes, the rises of the expert system are as a delivery system for extension information, to provide management education for decision makers and for dissemination of up-to-date scientific information in a readily accessible and easily understood form, to agricultural researchers, advisers and farmers. By the help of an expert system, the farmers can produce a more high quality product to the citizen.

### Expert System for Text Animation (ESTA)

The idea behind creating an expert system is that it can enable many people to benefit from the knowledge of one person – the expert. By providing it with a knowledge base for a certain subject area, ESTA can be used to create an expert system for the subject:

ESTA + Knowledge base = Expert System

Each knowledge base contains rules for a specific domain. A knowledge base for an expert system to give tax advice might contain rules relating marital status, mortgage commitments and age to the advisability of taking out a new life insurance policy. ESTA has all facilities to write the rules that will make up a knowledge base. ESTA has an inference engine which can use the rules in the knowledge base to determine which advice is to be given to the expert system user. ESTA also features the ability for the expert system user to obtain answers to questions such as “how” and “why”. ESTA is used by a knowledge engineer to create a knowledge base and by the expert system user to consult a knowledge base. Knowledge representation in ESTA is based on the items like sections, parameters, title.

CHAPTER-4

**KNOWLEDGE REPRESENTATION ISSUES**

# Knowledge Representation Representation and Mapping

* Problem solving requires large amount of knowledge and some mechanism for manipulating that knowledge.
* The Knowledge and the Representation are distinctentities, play a central but distinguishable roles in intelligent system.
  + **Knowledge** is a description of the world;

it determines a *system's competence* by what it knows.

* + **Representation** is the way knowledge is encoded;

it defines the *system's performance* in doing something.

* + **Facts** Truths about the real world and what we represent. This can be regarded as the knowledge level
* In simple words, we *:*
  + need to know about *things we want to represent* , and
  + need some means by which *things we can manipulate*.



Thus, knowledge representation can be considered at two levels :

* knowledge level at which facts are described, and
* symbol level at which the representations of the objects, defined in terms of symbols, can be manipulated in the programs.

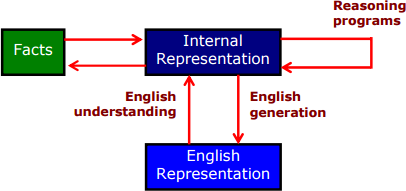
Note : A good representation enables fast and accurate access to knowledge and understanding of the content.

# Mapping between Facts and Representation

* Knowledge is a collection of “*facts”* from some domain.
* We need a representation of *"facts"* that can be manipulated by a program. Normal English is insufficient, too hard currently for a computer program to draw inferences in natural languages.
* Thus some symbolic representation is necessary.
* Therefore, we must be able to map *"facts to symbols"* and *"symbols to facts"* using

*forward and backward representation mapping.*

Example : Consider an English sentence

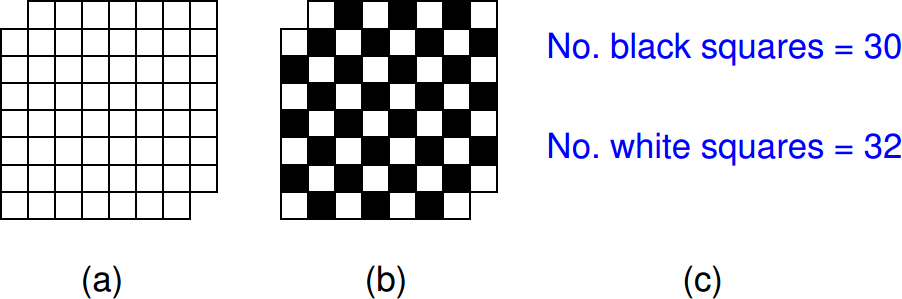




Now using deductive mechanism we can generate a new representation of object:

|  |  |
| --- | --- |
| Hastail (Spot) | A new object representation |
| Spot has a tail [it is new knowledge] | Using backward mapping function to generate English sentence |

* Good representation can make a reasoning program trivial
  + The Mutilated Checkerboard Problem: “Consider a normal checker board from which two squares, in opposite corners, have been removed. The task is to cover all the remaining squares exactly with dominoes, each of which covers two squares. No overlapping, either of dominoes on top of each other or of dominoes over the boundary of the mutilated board are allowed. Can this task be done?”



* Forward and Backward Representation

The forward and backward representations are elaborated below



* + The doted line on top indicates the abstract reasoning process that a program is intended to model.
  + The solid lines on bottom indicate the concrete reasoning process that the program performs.

# KR System Requirements

* A good knowledge representation enables fast and accurate access to knowledge and understanding of the content.
* A knowledge representation system should have following properties.

**Representational Adequacy** the ability to represent all kinds of knowledge that are needed in that domain.

**Inferential Adequacy** the ability to manipulate the representational structures to derive new structure corresponding to new knowledge inferred from old.

**Inferential Efficiency** the ability to incorporate additional information into the knowledge structure that can be used to focus attention of the inference mechanisms in the most promising direction.

**Acquisitional Efficiency** the ability to acquire new knowledge using automatic methods whenever possible rather than reliance on human intervention

**Note:** To date no single system can optimizes all of the above properties.

# Knowledge Representation Schemes

There are four types of Knowledge representation :

*Relational, Inheritable, Inferential, and Declarative/Procedural*.

# Relational Knowledge :

* + provides a framework to compare two objects based on equivalent attributes.
  + any instance in which two different objects are compared is a relational type of knowledge.

# Inheritable Knowledge

* + is obtained from associated objects.
  + it prescribes a structure in which new objects are created which may inherit all or a subset of attributes from existing objects.

# Inferential Knowledge

* + is inferred from objects through relations among objects.
  + e.g., a word alone is a simple syntax, but with the help of other words in phrase the reader may infer more from a word; this inference within linguistic is called semantics.

# Declarative Knowledge

* + a statement in which knowledge is specified, but the use to which that knowledge is to be put is not given.
  + e.g. laws, people's name; these are facts which can stand alone, not dependent on other knowledge;

# Procedural Knowledge

* + a representation in which the control information, to use the knowledge, is embedded in the knowledge itself.
  + e.g. computer programs, directions, and recipes; these indicate specific use or implementation;

# Relational Knowledge

This knowledge associates elements of one domain with another domain.

* Relational knowledge is made up of objects consisting of attributes and their corresponding associated values.
* The results of this knowledge type is a mapping of elements among different domains. The table below shows a simple way to store facts.
* The facts about a set of objects are put systematically in columns.
* This representation provides little opportunity for inference.

**Table - Simple Relational Knowledge**

|  |  |  |  |
| --- | --- | --- | --- |
| **Player** | **Height** | **Weight** | **Bats - Throws** |
| **Aaron** | **6-0** | **180** | **Right - Right** |
| **Mays** | **5-10** | **170** | **Right - Right** |
| **Ruth** | **6-2** | **215** | **Left - Left** |
| **Williams** | **6-3** | **205** | **Left - Right** |

* + Given the facts it is not possible to answer simple question such as :

## " Who is the heaviest player ? ".

but if a procedure for finding heaviest player is provided, then these facts will enable that procedure to compute an answer.

* + We can ask things like who "bats – left" and "throws – right".

# Inheritable Knowledge

* Here the knowledge elements inherit attributes from their parents.
* The knowledge is embodied in the design hierarchies found in the functional, physical and process domains. Within the hierarchy, elements inherit attributes from their parents, but in many cases not all attributes of the parent elements be prescribed to the child elements.
* The *inheritance* is a powerful form of inference, but not adequate. The basic KR needs to be augmented with inference mechanism.
* The KR in hierarchical structure, shown below, is called *“semantic network”* or a collection of *“frames” or “slot-and-filler structure".* The structure shows property inheritance and way for insertion of additional knowledge.
* Property inheritance: The objects or elements of specific classes inherit attributes and values from more general classes. The classes are organized in a generalized hierarchy.

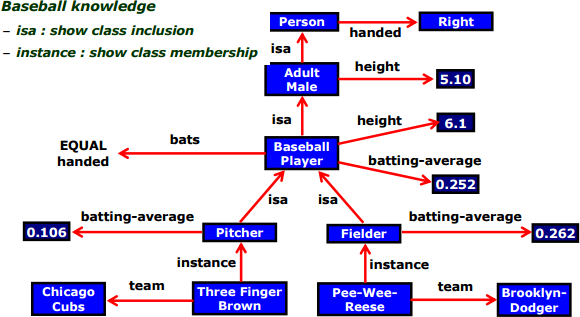


Fig. Inheritable knowledge representation (KR)

* + The directed arrows represent *attributes* (*isa, instance, team*) originates at object being described and terminates at object or its value.
  + The box nodes represents *objects* and *values* of the attributes.
* Viewing a node as a frame

Example : Baseball-player

isa : Adult-Male

Bates : EQUAL handed

Height : 6.1

Batting-average : 0.252

* Algorithm : Property Inheritance

Retrieve a value V for an attribute A of an instance object O Steps to follow:

1. Find object **O** in the knowledge base.
2. If there is a value for the attribute **A** then report that value.
3. Else, if there is a value for the attribute instance; If not, then fail.
4. Else, move to the node corresponding to that value and look for a value for the attribute **A**; If one is found, report it.
5. Else, do until there is no value for the “**isa**” attribute or until an answer is found :
   1. Get the value of the “**isa**” attribute and move to that node.
   2. See if there is a value for the attribute **A**; If yes, report it.

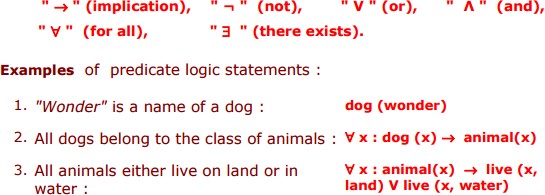
* This algorithm is simple. It describes the basic mechanism of inheritance. It does not say what to do if there is more than one value of the instance or “isa” attribute.
* This can be applied to the example of knowledge base illustrated, in the previous slide, to derive answers to the following queries :
  + team (Pee-Wee-Reese) = Brooklyn–Dodger
  + batting–average(Three-Finger-Brown) = 0.106
  + height (Pee-Wee-Reese) = 6.1
  + bats (Three Finger Brown) = right

# Inferential Knowledge

* This knowledge generates new information from the given information.
* This new information does not require further data gathering form source, but does require analysis of the given information to generate new knowledge.

Example :

* given a set of relations and values, one may infer other values or relations.
* a predicate logic (a mathematical deduction) is used to infer from a set of attributes.
* inference through predicate logic uses a set of logical operations to relate individual data.
* the symbols used for the logic operations are :



From these three statements we can infer that :

* Wonder lives either on land or on water."

Note : If more information is made available about these objects and their relations, then more knowledge can be inferred.

# Declarative/Procedural Knowledge

Differences between Declarative/Procedural knowledge is not very clear.

# Declarative knowledge :

Here, the knowledge is based on declarative facts about *axioms* and *domains* .

* + axioms are assumed to be true unless a counter example is found to invalidate them.
  + domains represent the physical world and the perceived functionality.
  + axiom and domains thus simply exists and serve as declarative statements that can stand alone.

# Procedural knowledge:

Here, the knowledge is a mapping process between domains that specify “what to do when” and the representation is of “how to make it” rather than “what it is”. The procedural knowledge :

* + may have inferential efficiency, but no inferential adequacy and acquisitional efficiency.
  + are represented as small programs that know how to do specific things, how to proceed.

Example : A parser in a natural language has the knowledge that a noun phrase may contain articles, adjectives and nouns. It thus accordingly call routines that know how to process articles, adjectives and nouns.

# Issues in Knowledge Representation

* The fundamental goal of Knowledge Representationis to facilitate inference (conclusions) from knowledge.
* The issues that arise while using KR techniques are many. Some of these are explained below.
  + Important Attributes :

Any attribute of objects so basic that they occur in almost every problem domain ?

* + Relationship among attributes:

Any important relationship that exists among object attributes ?

* + Choosing Granularity :

At what level of detail should the knowledge be represented ?

* + Set of objects :

How sets of objects be represented ?

* + Finding Right structure :

Given a large amount of knowledge stored, how can relevant parts be accessed ?

# Important Attributes

* + There are attributes that are of general significance.
  + There are two attributes "instance" and "isa", that are of general importance. These attributes are important because they support property inheritance.

# Relationship among Attributes

* + The attributes to describe objects are themselves entities they represent.
  + The relationship between the attributes of an object, independent of specific knowledge they encode, may hold properties like:
  + Inverses, existence in an isa hierarchy, techniques for reasoning about values and single valued attributes.

# Inverses :

This is about consistency check, while a value is added to one attribute. The entities are related to each other in many different ways. The figure shows attributes (isa, instance, and team), each with a directed arrow, originating at the object being described and terminating either at the object or its value.

There are two ways of realizing this:

* + - * first, represent two relationships in a single representation; e.g., a logical representation, team(Pee-Wee-Reese, Brooklyn–Dodgers), that can be interpreted as a statement about Pee-Wee-Reese or Brooklyn–Dodger.
      * second, use attributes that focus on a single entity but use them in pairs, one the inverse of the other; for e.g., one, team = Brooklyn– Dodgers , and the other, team = Pee-Wee-Reese, . . . .

This second approach is followed in semantic net and frame-based systems, accompanied by a knowledge acquisition tool that guarantees the consistency of inverse slot by checking, each time a value is added to one attribute then the corresponding value is added to the inverse.

* Existence in an "isa" hierarchy

This is about generalization-specialization, like, classes of objects and specialized subsets of those classes. There are attributes and specialization of attributes.

Example: the attribute "height" is a specialization of general attribute "physical-size" which is, in turn, a specialization of "physical-attribute". These generalization-specialization relationships for attributes are important because they support inheritance.

* Techniques for reasoning about values

This is about reasoning values of attributes not given explicitly. Several kinds of information are used in reasoning, like,

height : must be in a unit of length,

age : of person can not be greater than the age of person's parents.

The values are often specified when a knowledge base is created.

* Single valued attributes

This is about a specific attribute that is guaranteed to take a unique value.

Example : A baseball player can at time have only a single height and be a member of only one team. KR systems take different approaches to provide support for single valued attributes.

* Choosing Granularity

What level should the knowledge be represented and what are the primitives ?

* + Should there be a small number or should there be a large number of low-level primitives or High-level facts.
  + High-level facts may not be adequate for inference while Low-level primitives may require a lot of storage.

Example of Granularity :

* Suppose we are interested in following facts John spotted Sue.
* This could be represented as Spotted (agent(John), object (Sue))
* Such a representation would make it easy to answer questions such are Who spotted Sue ?
* Suppose we want to know Did John see Sue ?
* Given only one fact, we cannot discover that answer.
* We can add other facts, such as Spotted (x , y) -> saw (x , y)
* We can now infer the answer to the question.
* Set of Objects
  + Certain properties of objects that are true as member of a set but not as individual;

Example : Consider the assertion made in the sentences "there are more sheep than people in Australia", and "English speakers can be found all over the world."

* + To describe these facts, the only way is to attach assertion to the sets representing people, sheep, and English.
  + The reason to represent sets of objects is :

If a property is true for all or most elements of a set, then it is more efficient to associate it once with the set rather than to associate it explicitly with every elements of the set.

* + This is done in different ways :
* in logical representation through the use of universal quantifier, and
* in hierarchical structure where node represent sets, the inheritance propagate set level assertion down to individual.

Example: assert large (elephant); Remember to make clear distinction between,

* + whether we are asserting some property of the set itself, means, the set of elephants is large, or
  + asserting some property that holds for individual elements of the set , means, any thing that is an elephant is large.

There are three ways in which sets may be represented :

* + Name, as in the example. Inheritable KR, the node - Baseball- Player and the predicates as Ball and Batter in logical representation.
  + Extensional definition is to list the numbers, and
  + In tensional definition is to provide a rule, that returns true or false depending on whether the object is in the set or not.

# Finding Right Structure

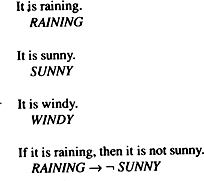
* + Access to right structure for describing a particular situation.
  + It requires, selecting an initial structure and then revising the choice. While doing so, it is necessary to solve following problems :
* how to perform an initial selection of the most appropriate structure.
* how to fill in appropriate details from the current situations.
* how to find a better structure if the one chosen initially turns out not to be appropriate.
* what to do if none of the available structures is appropriate.
* when to create and remember a new structure.
  + There is no good, general purpose method for solving all these problems. Some knowledge representation techniques solve some of them.

# CHAPTER 5 &6

**Knowledge Representation using predicate logic and Rules**

* 1. **Representing Simple Facts in Logic**

AI system might need to represent knowledge. Propositional logic is one of the fairly good forms of representing the same because it is simple to deal with and a decision procedure for it exists. Real-world facts are represented as logical propositions and are written as well-formed formulas (wff's) in propositional logic, as shown in Figure below. Using these propositions, we may easily conclude it is not sunny from the fact that its raining. But contrary to the ease of using the propositional logic there are its limitations. This is well demonstrated using a few simple sentence like:



Some simple facts in Propositional logic

Socrates is a man.

We could write:

*SOCRATESMAN*

But if we also wanted to represent Plato is a man.

we would have to write something such as:

*PLATOMAN*

which would be a totally separate assertion, and we would not be able to draw any conclusions about similarities between Socrates and Plato. It would be much better to represent these facts as:

*MAN(SOCRATES) MAN(PLATO)*

since now the structure of the representation reflects the structure of the knowledge itself. But to do that, we need to be able to use predicates applied to arguments. We are in even more difficulty if we try to represent the equally classic sentence

All men are mortal. We could represent this as:

*MORTALMAN*

But that fails to capture the relationship between any individual being a man and that individual being a mortal. To do that, we really need variables and quantification unless we are willing to write separate statements about the mortality of every known man.

Let's now explore the use of predicate logic as a way of representing knowledge by looking at a specific example. Consider the following set of sentences:

1 . Marcus was a man.

1. Marcus was a Pompeian.
2. All Pompeians were Romans.
3. Caesar was a ruler.
4. All Romans were either loyal to Caesar or hated him.
5. Everyone is loyal to someone.
6. People only try to assassinate rulers they are not loyal to.
7. Marcus tried to assassinate Caesar.

The facts described by these sentences can be represented as a set of wff's in predicate logic as follows:

1. Marcus was a man.

man(Marcus)

Although this representation fails to represent the notion of past tense (which is clear in the English sentence), it captures the critical fact of Marcus being a man. Whether this omission is acceptable or not depends on the use to which we intend to put the knowledge.

1. Marcus was a Pompeian. Pompeian(Marcus)
2. All Pompeians were Romans.

x : Pompeian(x) Roman(x) 4.Caesar was a ruler.

ruler(Caesar)

Since many people share the same name, the fact that proper names are often not references to unique individuals, overlooked here. Occasionally deciding which of several people of the same name is being referred to in a particular statement may require a somewhat more amount of knowledge and logic.

1. All Romans were either loyal to Caesar or hated him.

x: Roman(x) loyalto(x, Caesar) V hate(Caesar)

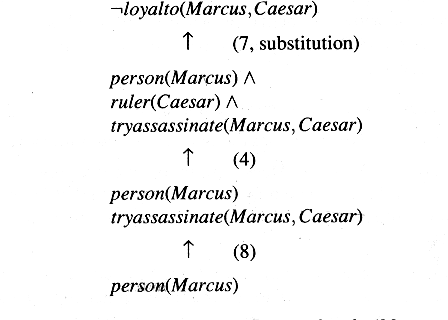
Here we have used the inclusive-or interpretation of the two types of Or supported by English language. Some people will argue, however, that this English sentence is really stating an exclusive-or. To express that. we would have to write:

x: Roman(x)  [(loyalto(x, Caesar) V hate(x, Caesar)) Not (loyalto(x, Caesar) hate(x, Caesar))]

1. Everyone is loyal to someone.

x:y : loyalto(x,y)

The scope of quantifiers is a major problem that arises when trying to convert English sentences into logical statements. Does this sentence say, as we have assumed in writing the logical formula above, that for each person there exists someone to whom he or she is loyal, possibly a different someone for everyone? Or does it say that there is someone to whom everyone is loyal?



An Attempt to Prove not loyal to(Marcus,Caesar)

1. People only try to assassinate rulers they are not loyal to.

x : y : person(x)  ruler(y)  tryassasinate(x,y)  loyalto(x,y)

1. Like the previous one this sentence too is ambiguous which may lead to more than one conclusion. The usage of “try to assassinate” as a single predicate gives us a fairly simple representation with which we can reason about trying to assassinate. But there might be connections as try to assassinate and not actually assassinate could not be made easily.
2. Marcus tried to assassinate Caesar. tryassasinate (Marcus,Caesar)

now, say suppose we wish to answer the following question:

Was Marcus loyal to Caesar?

What we do is start reasoning backward from the desired goal which is represented in predicate logic as:

loyalto(Marcus, Caesar)

Figure 4.2 shows an attempt to produce a proof of the goal by reducing the set of necessary but as yet unattained goals to the empty sets. The attempts fail as we do not have any statement to prove person(Marcus). But the problem is solved just by adding an additional statement i.e.

1. All men are people.

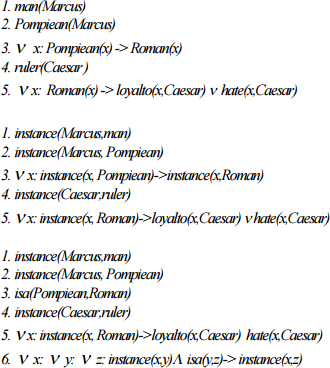
x : man(x)  person(x)

Now we can satisfy the last goal and produce a proof that Marcus was not loyal to Caesar.

# Representing Instance and isa relationships

* + - * Knowledge can be represented as classes, objects, attributes and Super class and sub class relationships.
      * Knowledge can be inference using property inheritance. In this elements of specific classes inherit the attributes and values.
      * Attribute instance is used to represent the relationship “Class membership ” (element of the class)
      * Attribute isa is used to represent the relationship “Class inclusion” (super class, sub class relationship)

Three ways of representing class membership



These examples illustrate two points. The first is fairly specific. It is that, although class and superclass memberships are important facts that need to be represented those memberships need not be represented with predicates labelled instance and isa. In fact, in a logical framework it is usually unwieldy to do that, and instead unary predicates corresponding to the classes are often used. The second point is more general. There are usually several different ways of representing a given fact within a particular representational framework, be it logic or anything else. The choice depends partly on which deductions need to be supported most efficiently and partly on taste. The only important thing is that within a particular knowledge base consistency of representation is critical. Since any particular inference rule is designed to work on one particular form of representation, it is necessary that all the knowledge to which that rule is intended to apply be in the form that the rule demands. Many errors in the reasoning performed by knowledge-based programs are the result of inconsistent representation decisions. The moral is simply to be careful.

# Computable functions and predicates

* + - * Some of the computational predicates like Less than, Greater than used in knowledge representation.
      * It generally return true or false for the inputs.

Examples: Computable predicates

gt(1,0) or lt(0,1)

gt(5,4) or gt(4,5)

Computable functions: gt(2+4, 5)

Consider the following set of facts, again involving Marcus:

1. marcus was a man man(Marcus)
2. Marcus was a pompeian Pompeian(Marcus)
3. Marcus was born in 40 A.D born(marcus, 40)
4. All men are mortal

∀x: men(x)→ mortal(x)

1. All Pompeians died when the volcano erupted in 79 A.D erupted(volcano,79) & x :pompeian(x)→died(x, 79)
2. No mortal lives longer than150 years

∀x: ∀t1: ∀t2: mortal(x) & born(x,t1) & gt(t2-t1,150)→ dead(x,t1)

1. It is Now 1991 Now=1991
2. Alive means not dead

∀x: ∀t: [ alive(x,t) →~dead(x,t)] & [~dead(x,t)→alive(x,t)]

1. If someone dies then he is dead at all later times

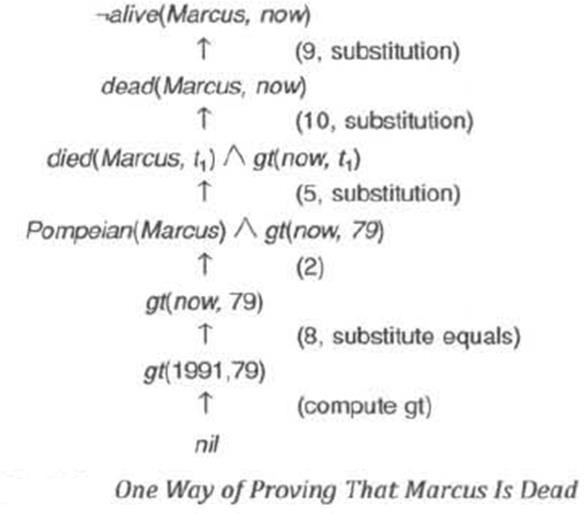
∀x: ∀t1: ∀t2: died(x,t1) & gt(t2,t1)→ dead(x1,t2)

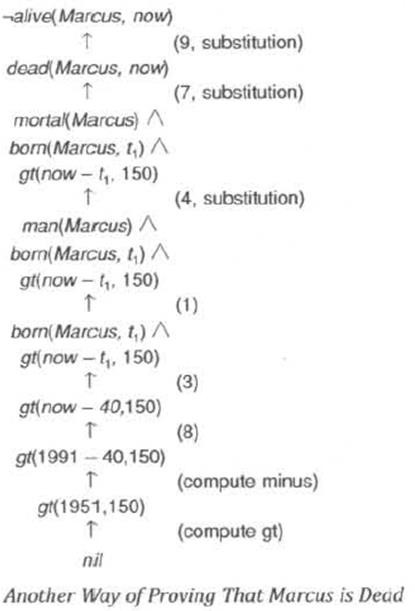
This representation says that one is dead in all years after the one in which one died. It ignores the question of whether one is dead in the year in which one died.

1. man(Marcus)
2. Pompeian(Marcus)
3. born(marcus, 40)
4. ∀x: men(x)→ mortal(x)
5. ∀:pompeian(x)→died(x, 79)
6. erupted(volcano,79)

7. ∀ x: ∀t1: ∀t2: mortal(x) & born(x,t1) & gt(t2-t1,150)→ dead(x,t1)

8. Now=1991

9. ∀x: ∀ t: [ alive(x,t) →~dead(x,t)] &[~dead(x,t)→alive(x,t)] 10. ∀ x: ∀t1: ∀t2: died(x,t1) & gt(t2,t1)→ dead(x1,t2)



Two things should be clear from the proofs we have just shown:

* Even very simple conclusions can require many steps to prove.
* A variety of processes, such as matching, substitution, and application of modus ponens are involved in the production of a proof, This is true even for the simple statements we are using, It would be worse if we had implications with more than a single term on the right or with complicated expressions involving ands and ors on the left.

# Disadvantage:

* Many steps required to prove simple conclusions
* Variety of processes such as matching and substitution used to prove simple conclusions

# Resolution

* Resolution is a proof procedure by refutation.
* To prove a statement using resolution it attempt to show that the negation of that statement.

## Algorithm: Convert to Clause Form

1. Eliminate →, using the fact that a → b is equivalent to ¬ a V b. Performing this transformation on the wff given above yields

∀x: ¬ [Roman(x) ∧ know(x, Marcus)] V

[hate(x, Caesar) V (∀y : ¬(∃z : hate(y, z)) V thinkcrazy(x, y))]

1. Reduce the scope of each ¬ to a single term, using the fact that ¬ (¬ p) = p, deMorgan's laws [which say that ¬ (a ∧ b) = ¬ a V ¬ b and ¬ (a V b) = ¬ a ∧ ¬ b ], and the standard correspondences between quantifiers [¬ ∀x: P(x) = ∃x: ¬ P(x) and ¬ ∃x: P(x) = ∀ x: ¬P(x)]. Performing this transformation on the wff from step 1 yields

∀x: [¬ Roman(x) V ¬ know(x, Marcus)] V

[hate(x, Caesar) V (∀y: ∀z: ¬ hate(y, z) V thinkcrazy(x, y))]

1. Standardize variables so that each quantifier binds a unique variable. Since variables are just dummy names, this process cannot affect the truth value of the wff. For example, the formula

∀x: P(x) V ∀x: Q(x) would be converted to

∀x: P(x) V ∀y: Q(y)

1. Move all quantifiers to the left of the formula without changing their relative order. This is possible since there is no conflict among variable names. Performing this operation on the formula of step 2, we get

∀x: ∀y: Az: [¬ Roman(x) V ¬ know(x, Marcus)] V [hate(x, Caesar) V (¬ hale(y, z) V thinkcrazy(x, y))]

At this point, the formula is in what is known as prenex normal form. It consists of a prefix of quantifiers followed by a matrix, which is quantifier-free.

1. Eliminate existential quantifiers. A formula that contains an existentially quantified variable asserts that there is a value that can be substituted for the variable that makes the formula true. We can eliminate the quantifier by substituting for the variable a reference to a function that produces the desired value. Since we do not necessarily know how to produce the value, we must create a new function name for every such replacement. We make no assertions about these functions except that they must exist. So, for example, the formula

∃y : President(y)

can be transformed into the formula President(S1)

where S1 is a function with no arguments that somehow produces a value that satisfies President. If existential quantifiers occur within the scope of universal quantifiers, then the value that satisfies the predicate may depend on the values of the universally quantified variables. For example, in the formula

∀x: ∃y: father-of(y, x)

the value of y that satisfies father-of depends on the particular value of x. Thus we must generate functions with the same number of arguments as the number of universal quantifiers in whose scope the expression occurs. So this example would be transformed into

∀x: father-of(S2(x), x)

These generated functions are called Skolem functions. Sometimes ones with no arguments are called Skolem constants.

1. Drop the prefix. At this point, all remaining variables are universally quantified, so the prefix can just be dropped and any proof procedure we use can simply assume that any variable it sees is universally quantified. Now the formula produced in step 4 appears as

[¬ Roman(x) V ¬ know(x, Marcus)] V

[hate(x, Caesar) V (¬ hate(y, z) V thinkcrazy(x, y))]

1. Convert the matrix into a conjunction of disjuncts. In the case or our example, since there are no and‟s, it is only necessary to exploit the associative property of or [ i.e., (a ∧ b) V c = (a V c)

∧ (b ∧ c)] and simply remove the parentheses, giving

¬ Roman(x) V ¬ know(x, Marcus) V

hate(x, Caesar) V ¬ hate(y, z) V thinkcrazy(x, y)

However, it is also frequently necessary to exploit the distributive property [i.e. , (a ∧ b) V c = (a V c) ∧ (b V c)]. For example, the formula

(winter ∧ wearingboots) V (summer ∧ wearingsandals)

Becomes, after one application of the rule

[winter V (summer ∧ wearingsandals)]

∧ [wearingboots V (summer ∧ wearingsandals)]

And then, after a second application, required since there are still conjuncts joined by OR's,

(winter V summer) ∧ (winter V wearingsandals) ∧ (wearingboots V summer) ∧

(wearingboots V wearingsandals)

1. Create a separate clause corresponding to each conjunct. In order for a wff to be true, all the clauses that are generated from it must be true. If we are going to be working with several wff‟s, all the clauses generated by each of them can now be combined to represent the same set of facts as were represented by the original wff's.
2. Standardize apart the variables in the set of clauses generated in step 8. By this we mean rename the variables so that no two clauses make reference to the same variable. In making this transformation, we rely on the fact that

*(*∀x: *P(x)* ∧ *Q(x)) =* ∀x: *P(x)* ∧ ∀x: *Q(x)*

Thus since each clause is a separate conjunct and since all the variables are universally quantified, there need be no relationship between the variables of two clauses, even if they were generated from the same wff.

Performing this final step of standardization is important because during the resolution procedure it is sometimes necessary to instantiate a universally quantified variable (i.e., substitute for it a particular value). But, in general, we want to keep clauses in their most general form as long as possible. So when a variable is instantiated, we want to know the minimum number of substitutions that must be made to preserve the truth value of the system.

After applying this entire procedure to a set of wff's, we will have a set of clauses, each of which is a disjunction of *literals.* These clauses can now be exploited by the resolution procedure to generate proofs.

# 2.5.1 Resolution in Propositional Logic

In propositional logic, the procedure for producing a proof by resolution of proposition P with respect to a set of axioms F is the following.

ALGORITHM: PROPOSITIONAL RESOLUTION

1. Convert all the propositions of F to clause form
2. Negate P and convert the result to clause form. Add it to the set of clauses obtained in step 1.
3. Repeat until either a contradiction is found or no progress can be made:
4. Select two clauses. Call these the parent clauses.
5. Resolve them together. The resulting clause, called the resolvent, will be the disjunction of all of the literals of both of the parent clauses with the following exception: If there are any pairs of literals L and ¬L such that one of the parent clauses contains L and the other contains ¬L, then select one such pair and eliminate both L and ¬L from the resolvent.
6. If the resolvent is the empty clause, then a contradiction has been found. If it is not then add it to the set of clauses available to the procedure.

Suppose we are given the axioms shown in the first column of Table 1 and we want to prove R.First we convert the axioms to clause which is already in clause form. Then we begin selecting pairs of clauses to resolve together. Although any pair of clauses can be resolved, only those pairs that contain complementary literals will produce a resolvent that is likely to lead to the goal of sequence of resolvents shown in figure 1. We begin by resolving with the clause ⱷR since that is one of the clauses that must be involved in the contradiction we are trying to find.

One way of viewing the resolution process is that it takes a set of clauses that are all assumed to be true and based on information provided by the others, it generates new clauses that represent restrictions on the way each of those original clauses can be made true. A contradiction occurs when a clause becomes so restricted that there is no way it can be true. This is indicated by the generation of the empty clause.

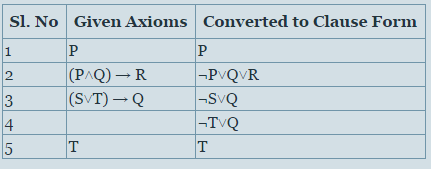


Table 1 Some Facts in Propositional Logic

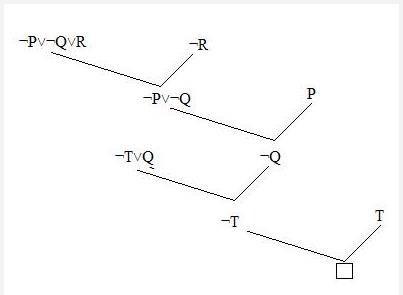


Figure : Resolution In Propositional Logic

# UNIFICATION ALGORITHM

In propsoitional logic it is easy to determine that two literals can not both be true at the same time. Simply look for L and ~L . In predicate logic, this matching process is more complicated, since bindings of variables must be considered.

For example man (john) and man(john) is a contradiction while man (john) and man(Himalayas) is not. Thus in order to determine contradictions we need a matching procedure that compares two literals and discovers whether there exist a set of substitutions that makes them identical . There is a recursive procedure that does this matching . It is called Unification algorithm.

In Unification algorithm each literal is represented as a list, where first element is the name of a predicate and the remaining elements are arguments. The argument may be a single element (atom) or may be another list. For example we can have literals as

( tryassassinate Marcus Caesar)

( tryassassinate Marcus (ruler of Rome))

To unify two literals , first check if their first elements re same. If so proceed. Otherwise they can not be unified. For example the literals

( try assassinate Marcus Caesar)

( hate Marcus Caesar)

Can not be Unfied. The unification algorithm recursively matches pairs of elements, one pair at a time. The matching rules are :

1. Different constants , functions or predicates can not match, whereas identical ones can.
2. A variable can match another variable , any constant or a function or predicate expression, subject to the condition that the function or [predicate expression must not contain any instance of the variable being matched (otherwise it will lead to infinite recursion).
3. The substitution must be consistent. Substituting y for x now and then z for x later is inconsistent. (a substitution y for x written as y/x)

The Unification algorithm is listed below as a procedure UNIFY (L1, L2). It returns a list representing the composition of the substitutions that were performed during the match. An empty list NIL indicates that a match was found without any substitutions. If the list contains a single value F, it indicates that the unification procedure failed.

The empty list, NIL, indicates that a match was found without any substitutions. The list consisting of the single value FAIL indicates that the unification procedure failed.

Algorithm: Unify(L1, L2)

1. If L1 or L2 are both variables or constants, then:
   1. If L1 and L2 are identical, then return NIL.
   2. Else if L1 is a variable, then if L1 occurs in L2 then return {FAIL}, else return (L2/L1).
   3. Else if L2 is a variable, then if L2 occurs in L1 then return {FAIL} , else return (L1/L2).
   4. Else return {FAIL}.
2. If the initial predicate symbols in L1 and L2 are not identical, then return {FAIL}.
3. If LI and L2 have a different number of arguments, then return {FAIL}.
4. Set SUBST to NIL. (At the end of this procedure, SUBST will contain all the substitutions used to unify L1 and L2.)
5. For i ← 1 to number of arguments in L1 :
   1. Call Unify with the ith argument of L1 and the ith argument of L2, putting result in S.
   2. If S contains FAIL then return {FAIL}.
   3. If S is not equal to NIL then:
      1. Apply S to the remainder of both L1 and L2.
      2. SUBST: = APPEND(S, SUBST).
6. Return SUBST.

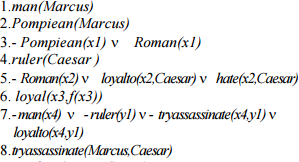
# Resolution in Predicate Logic

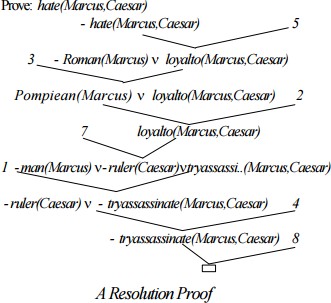
**ALGORITHM: RESOLUTION IN PREDICATE LOGIC**

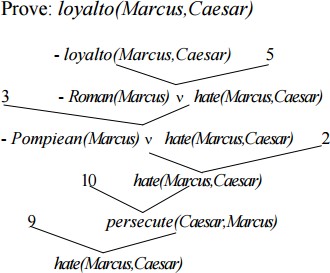
1. Convert all the statements of F to clause form
2. Negate P and convert the result to clause form. Add it to the set of clauses obtained in step 1.
3. Repeat until either a contradiction is found or no progress can be made or a predetermined amount of effort has been expended:
4. Select two clauses. Call these the parent clauses.
5. Resolve them together. The resulting clause, called the resolvent, will be the disjunction of all of the literals of both the parent clauses with appropriate substitutions performed and with the following exception: If there is one pair of literals T1 and T2 such that one of the parent clauses contains T1 and the other contains T2 and if T1 and T2 are unifiable, then neither T1 nor T2 should appear in the resolvent. We call T1 and T2 complementary literals. Use the substitution produced by the unification to create the resolvent. If there is one pair of complementary literals, only one such pair should be omitted from the resolvent.
6. If the resolvent is the empty clause, then a contradiction has been found. If it is not then add it to the set of clauses available to the procedure.

If the choice of clauses to resolve together at each step is made in certain systematic ways, then the resolution procedure will find a contradiction if one exists. However, it may take a very long time. There exist strategies for making the choice that can speed up the process considerably as given below.

# Axioms in clause form

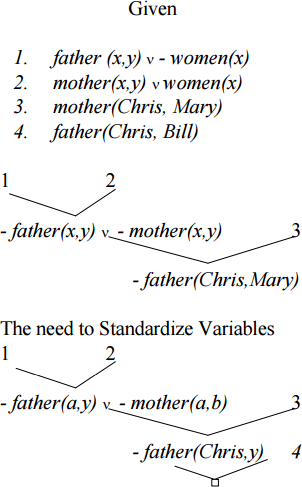






An Unsuccessful Attempt at Resolution

1. persecute(x, y)  hate(y, x)
2. hate(x,y)  persecute(y, x) Converting to clause form, we get
3.  persecute(x5,y2)  hate(y2,x5)
4. hate(x6,y3)  persecute(y3,x6)



# Procedural v/s Declarative Knowledge

* + - * A Declarative representation is one in which knowledge is specified but the use to which that knowledge is to be put in, is not given.
      * A Procedural representation is one in which the control information that is necessary to use the knowledge is considered to be embedded in the knowledge itself.
      * To use a procedural representation, we need to augment it with aninterpreter that follows the instructions given in the knowledge.
      * The difference between the declarative and the procedural views of knowledge lies in where control information resides.

man(Marcus) man(Caesar) person(Cleopatra)

∀x: man(x)→ person(x)

person(x)?

Now we want to extract from this knowledge base the ans to the question:

Ǝy : person (y)

Marcus, Ceaser and Cleopatra can be the answers

* + - * As there is more than one value that satisfies the predicate, but only one value is needed, the answer depends on the order in which the assertions are examined during the search of a response.
      * If we view the assertions as declarative, then we cannot depict how they will be examined. If we view them as procedural, then they do.
      * Let us view these assertions as a non deterministic program whoseoutput is simply not defined, now this means that there is no difference between Procedural & Declarative Statements. But most of the machines don‟t do so, they hold on to what ever method they have, either sequential or in parallel.
      * The focus is on working on the control model. man(Marcus)

man (Ceaser) Vx : man(x) person(x)

Person(Cleopatra)

* + - * If we view this as declarative then there is no difference with the previous statement. But viewed procedurally, and using the control model, we used to got Cleopatra as the answer, now the answer is marcus.
      * The answer can vary by changing the way the interpreter works.
      * The distinction between the two forms is often very fuzzy. Rather then trying to prove which technique is better, what we should do is to figure out what the ways in which rule formalisms and interpreters can be combined to solve problems.

# Logic Programming

* + - * Logic programming is a programming language paradigm in whichlogical assertions are viewed as programs, e.g : PROLOG
      * A PROLOG program is described as a series of logical assertions, eachof which is a Horn Clause.
      * A Horn Clause is a clause that has at most one positive literal.
      * Eg p, ¬ p V q etc are also Horn Clauses.
      * The fact that PROLOG programs are composed only of Horn Clauses and not of arbitrary logical expressions has two important consequences.
      * Because of uniform representation a simple & effective interpreter can be written.
      * The logic of Horn Clause systems is decidable.
      * Even PROLOG works on backward reasoning.
      * The program is read top to bottom, left to right and search is performed depth-first with backtracking.
      * There are some syntactic difference between the logic and the PROLOG representations as mentioned
      * The key difference between the logic & PROLOG representation is that PROLOG interpreter has a fixed control strategy, so assertions in the PROLOG program define a particular search path to answer any question. Whereas Logical assertions define set of answers that they justify, there can be more than one answers, it can be forward or backward tracking.
      * Control Strategy for PROLOG states that we begin with a problem statement, which is viewed as a goal to be proved.
      * Look for the assertions that can prove the goal.
      * To decide whether a fact or a rule can be applied to the current problem, invoke a standard unification procedure.
      * Reason backward from that goal until a path is found that terminates with assertions in the program.
      * Consider paths using a depth-first search strategy and use backtracking.
      * Propagate to the answer by satisfying the conditions.

# Forward v/s Backward Reasoning

* + - * The objective of any search is to find a path through a problem spacefrom the initial to the final one.
      * There are 2 directions to go and find the answer
        + Forward
        + Backward
      * 8-square problem
      * Reason forward from the initial states: Begin building a tree of move sequences that might be solution by starting with the initialconfiguration(s) at the root of the tree. Generate the next level of tree by finding all the rules whose left sides match the root node and use the right sides to create the new configurations. Generate each node by taking each node generated at the previous level and applying to it all of the rules whose left sides match it. Continue.
      * Reason backward from the goal states: Begin building a tree of move sequences that might be solution by starting with the goal configuration(s) at the root of the tree. Generate the next level of tree by finding all the rules whose right sides match the root node and use the left sides to create the new configurations. Generate each node by taking each node generated at the previous level and applying to it all of the rules whose right sides match it. Continue. This is also called Goal-Directed Reasoning.
      * To summarize, to reason forward, the left sides (pre-conditions) are matched against the current state and the right sides (the results) are used to generate new nodes until the goal is reached.
      * To reason backwards, the right sides are matched against the current node and the left sides are used to generate new nodes.
      * Factors that influence whether to choose forward or backward reasoning:
        + Are there more possible start states or goal states? We would like to go from smaller set of states to larger set of states.
        + In which direction is the branching factor (the average number of nodes that can be reached directly from a single node) greater? We would like to proceed in the direction with the lower branching factor.
        + Will the program be asked to justify its reasoning process to the user? It so, it is important to proceed in the direction that corresponds more closely with the way user will think.
        + What kind of event is going to trigger a problem-solving episode? If it is the arrival of a new fact , forward reasoning should be used. If it a query to which response is desired, use backward reasoning.
      * Home to unknown place example.
      * MYCIN
      * Bidirectional Search (The two searches must pass each other)
      * Forward Rules: which encode knowledge about how to respond to certain input configurations.
      * Backward Rules: which encode knowledge about how to achieve particular goals.
      * Backward- Chaining Rule Systems
        + PROLOG is an example of this.
        + These are good for goal-directed problem solving.
        + Hence Prolog & MYCIN are examples of the same.
      * Forward - Chaining Rule Systems
        + We work on the incoming data here.
        + The left sides of rules are matched with against the state description.
        + The rules that match the state dump their right side assertions into the state.
        + Matching is more complex for forward chaining systems.
        + OPS5, Brownston etc. are the examples of the same.
      * Combining Forward v/s Backward Reasoning
        + Patients example of diagnosis.
        + In some systems, this is only possible in reversible rules.

# Matching

* + - * Till now we have used search to solve the problems as the application of appropriate rules.
      * We applied them to individual problem states to generate new states to which the rules can then be applied, until a solution is found.
      * We suggest that a clever search involves choosing from among the rules that can be applied at a particular point, but we do not talk about how to extract from the entire collection of rules those that can be applied at a given point.
      * To do this we need matching.

# Indexing

* + - * Do a simple search through all the rules, comparing each one‟s precondition to the current state and extracting all the ones that match.
      * But this has two problems
      * In order to solve very interesting problems, it will be necessary to use a large number of rules, scanning through all of them at every step of the search would be hopelessly inefficient.
      * It is not always immediately obvious whether a rule‟s preconditions are satisfied by a particular state.
      * To solve the first problem, use simple indexing. E.g. in Chess, combine all moves at a particular board state together.

# Matching with Variables

* + - * The problem of selecting applicable rules is made more difficult when preconditions are not stated as exact descriptions of particular situations but rather describe properties that the situations must have.
      * Then we need to match a particular situation and the preconditions of a given situation.
      * In many rules based systems, we need to compute the whole set of rules that match the current state description. Backward Chaining Systems usually use depth-first backtracking to select individual rules, but forward chaining systems use Conflict Resolution Strategies.
      * One efficient many to many match algorithm is RETE

Complex &Approximate Matching

* + - * A more complex matching process is required when the preconditions of a rule specify required properties that are not stated explicitly in thedescription of the current state. In this case, a separate set of rules must be used to describe how some properties can be inferred from others.
      * An even more complex matching process is required if rules should be applied if their preconditions approximately match the current situation. Example of listening to a recording of a telephonic conversation.
      * For some problems, almost all the action is in the matching of the rulesto the problem state. Once that is done, so few rules apply that theremaining search is trivial. Example ELIZA

# Conflict Resolution

* + - * The result of the matching process is a list of rules whose antecedents have matched the current state description along with whatever variable binding were generated by the matching process.
      * It is the job of the search method to decide on the order in which the rules will be applied. But sometimes it is useful to incorporate some of the decision making into the matching process. This phase is called conflict resolution.
      * There are three basic approaches to the problem of conflict resolution in the production system
        + Assign a preference based on the rule that matched.
        + Assign a preference based on the objects that matched.
        + Assign a preference based on the action that the matched rule would perform.

STRUCTURED REPRESNTATION OF KNOWLEDGE

* + - * Representing knowledge using logical formalism, like predicate logic, has several advantages. They can be combined with powerful inference mechanisms like resolution, which makes reasoning with facts easy. But using logical formalism complex structures of the world, objects and their relationships, events, sequences of events etc. cannot be described easily.
      * A good system for the representation of structured knowledge in a particular domain should posses the following four properties:

1. Representational Adequacy:- The ability to represent all kinds of knowledge that are needed in that domain.
2. Inferential Adequacy :- The ability to manipulate the represented structure and infer new structures.
3. Inferential Efficiency:- The ability to incorporate additional information into the knowledge structure that will aid the inference mechanisms.
4. Acquisitional Efficiency :- The ability to acquire new information easily, either by direct insertion or by program control.
   * + - The techniques that have been developed in AI systems to accomplish these objectives fall under two categories:
5. Declarative Methods:- In these knowledge is represented as static collection of facts which are manipulated by general procedures. Here the facts need to be stored only one and they can be used in any number of ways. Facts can be easily added to declarative systems without changing the general procedures.
6. Procedural Method:- In these knowledge is represented as procedures. Default reasoning and probabilistic reasoning are examples of procedural methods. In these, heuristic knowledge of “How to do things efficiently “can be easily represented.
   * + - In practice most of the knowledge representation employ a combination of both. Most of the knowledge representation structures have been developed to handle programs that handle natural language input. One of the reasons that knowledge structures are so important is that they provide a way to represent information about commonly occurring patterns of things . such descriptions are some times called schema. One definition of schema is
       - “Schema refers to an active organization of the past reactions, or of past experience, which must always be supposed to be operating in any well adapted organic response”.
       - By using schemas, people as well as programs can exploit the fact that the real world is not random. There are several types of schemas that have proved useful in AI programs.

They include

1. Frames:- Used to describe a collection of attributes that a given object possesses (eg: description of a chair).
2. Scripts:- Used to describe common sequence of event (eg:- a restaurant scene).
3. Stereotypes :- Used to described characteristics of people.
4. Rule models:- Used to describe common features shared among a set of rules in a production system.
   * + - Frames and scripts are used very extensively in a variety of AI programs. Before selecting any specific knowledge representation structure, the following issues have to be considered.
5. The basis properties of objects , if any, which are common to every problem domain must be identified and handled appropriately.
6. The entire knowledge should be represented as a good set of primitives.
7. Mechanisms must be devised to access relevant parts in a large knowledge base.

**CHAPTER-12**

**GAME PLAYING**

Game Playing is one of the oldest sub-fields in AI. Game playing involves abstract and pure form of competition that seems to require intelligence. It is easy to represent the states and actions. To implement the game playing very little world knowledge is required.

The most common used AI technique in game is search. Game playing research has contributed ideas on how to make the best use of time to reach good decisions.

Game playing is a search problem defined by:

* + - Initial state of the game
    - Operators defining legal moves
    - Successor function
    - Terminal test defining end of game states
    - Goal test
    - Path cost/utility/payoff function

More popular games are too complex to solve, requiring the program to take its best guess. “ for example in chess, the search tree has 1040 nodes (with branching factor of 35). It is the opponent because of whom uncertainty arises.

Characteristics of game playing

1. There are always an “unpredictable” opponent:
   * The opponent introduces uncertainty
   * The opponent also wants to win

The solution for this problem is a strategy, which specifies a move for every possible opponent reply.

1. Time limits:

Game are often played under strict time constraints (eg:chess) and therefore must be very effectively handled.

There are special games where two players have exactly opposite goals. There are also perfect information games(sch as chess and go) where both the players have access to the same information about the game in progress (e.g. tic-tac-toe). In imoerfect game information games (such as bridge or certain card games and games where dice is used). Given sufficient time and space, usually an optimum solution can be obtained for the former by exhaustive search, though not for the latter.

# Types of games

There are basically two types of games

* Deterministic games
* Chance games

Game like chess and checker are perfect information deterministic games whereas games like scrabble and bridge are imperfect information. We will consider only two player discrete, perfect information games, such as tic-tac-toe, chess, checkers etc... . Two- player games are easier to imagine and think and more common to play.

# Minimize search procedure

Typical characteristic of the games is to look ahead at future position in order to succeed. There is a natural correspondence between such games and state space problems.

In a game like tic-tac-toe

* States-legal board positions
* Operators-legal moves
* Goal-winning position

The game starts from a specified initial state and ends in position that can be declared win for one player and loss for other or possibly a draw. Game tree is an explicit representation of all possible plays of the game. We start with a 3 by 3 grid..



Then the two players take it in turns to place a there marker on the board( one player uses the „X‟ marker, the other uses the „O‟ marker). The winner is the player who gets 3 of these markers in a row, eg.. if X wins



Another possibility is that no1 wins eg..



Or the third possibility is a draw case

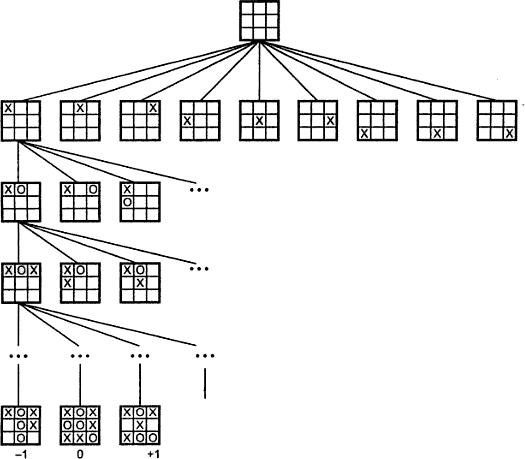


# Search tree for tic-tac-toe

The root node is an initial position of the game. Its successors are the positions that the first player can reach in one move; their successors are the positions resulting from the second player's replies and so on. Terminal or leaf nodes are presented by WIN, LOSS or DRAW. Each path from the root ro a terminal node represents a different complete play of the game. The moves available to one player from a given position can be represented by OR links whereas the moves available to his opponent are AND links.

The trees representing games contain two types of nodes:

* + MAX- nodes (assume at even level from root)
  + MIN - nodes [assume at odd level from root)



Search tree for tic-tac-toe

the leaves nodes are labeled WIN, LOSS or DRAW depending on whether they represent a win, loss or draw position from Max‟s viewpoint. Once the leaf nodes are assigned their WIN-LOSS or DRAW status, each nodes in the game tree can be labeled WIN, LOSS or DRAW by a bottom up process.

Game playing is a special type of search, where the intention of all players must be taken into account.

# Minimax procedure

* Starting from the leaves of the tree (with final scores with respect to one player, MAX), and go backwards towards the root.
* At each step, one player (MAX) takes the action that leads to the highest score, while the other player(MIN) takes the action that leads to the lowest score.
* All the nodes in the tree will be scored and the path from root to the actual result is the one on which all node have the same score.

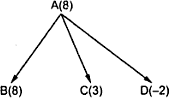
The minimax procedure operates on a game tree and is recursive procedure where a player tries to minimize its opponent‟s advantage while at the same time maximize its own. The player hoping for positive number is called the maximizing player. His opponent is the minimizing player. If the player to move is the maximizing player, he is looking for a path leading to a large positive number and his opponent will try to force the play toward situation with strongly

negative static evaluations. In game playing first construct the tree up till the depth-bound and then compute the evaluation function for the leaves. The next step is to propagate the values up to the starting.

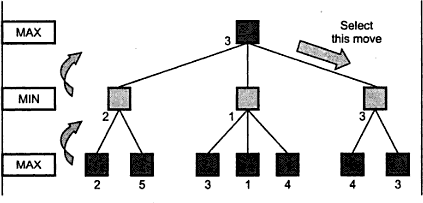
The procedure by which the scoring information passes up the game tree is called the MINIMAX procedures since the score at each node is either minimum or maximum of the scores at the nodes immediately below

# One-ply search

In this fig since it is the maximizing search ply 8 is transferred upwards to A



# Two-ply search



**Static evaluation function**

To play an entire game we need to combine search oriented and non-search oriented techniques. The idea way to use a search procedure to find a solution to the problem statement is to generate moves through the problem space until a goal state is reached. Unfortunately for games like

chess even with a good plausible move generator, it is not possible to search until goal state is reached. In the amount of time available it is possible to generate the tree at the most 10 to 20 ply deep. Then in order to choose the best move, the resulting board positions must be compared to discover which is most advantageous. This is done using the static evaluation function. The static evaluation function evaluates individual board positions by estimating how much likely they are eventually to lead to a win.

The minimax procedure is a depth-first, depth limited search procedure.

* If the limit of search has reached, compute the static value of the current position relative to the appropriate layer as given below (maximizing or minimizing player). Report the result (value and path).
* If the level is minimizing level(minimizer‟s turn)
* Generate the successors of the current position. Apply MINIMAX to each of the successors. Return the minimum of the result.
* If the level is a maximizing level. Generate the successors of current position Apply MINIMAX to each of these successors. Return the maximum of the result. The maximum algorithm uses the following procedures

1. MOVEGEN(POS)

It is plausible move generator. It returns a list of successors of „Pos‟.

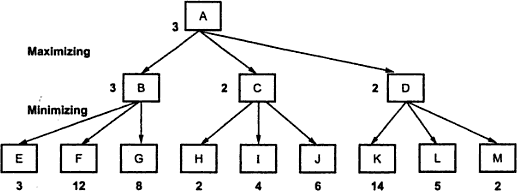
1. STSTIC (Pos, Depth)

The static evaluation function that returns a number representing the goodness of „pos‟ from the current point of view.

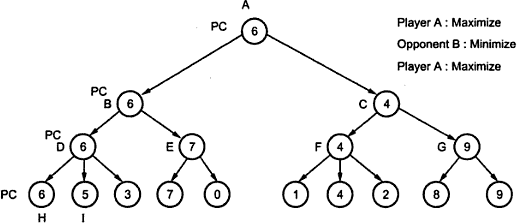
1. DEEP-ENOUGH

It returns true if the search to be stopped at the current level else it returns false.

A MINIMAX example



Another example of minimax search procedure



In the above example, a Minimax search in a game tree is simulated. Every leaf has a corresponding value, which is approximated from player A‟s view point. When a path is chosen, the value of the child will be passed back to the parent. For example, the value for D is 6, which is the maximum value of its children, while the value for C is 4 which is the minimum value of F and G. In this example the best sequence of moves found by the maximizing/minimizing procedure is the path through nodes A, B, D and H, which is called the principal continuation. The nodes on the path are denoted as PC (principal continuation) nodes. For simplicity we can modify the game tree values slightly and use only maximization operations. The trick is to maximize the scores by negating the returned values from the children instead of searching for minimum scores and estimate the values at leaves from the player‟s own viewpoin

PART – A

1. How is predicate logic helpful in knowledge representation?
2. Define semantic networks.
3. What is the need of facts and its representation?
4. What is property inheritance?
5. Discuss in brief about ISA and Instance classes.
6. Give some use of conceptual dependency.
7. Define inference.
8. Define logic.
9. Write short notes on uniqueness quantifier.
10. Write short notes on uniqueness operator.
11. Define WWF with an example.
12. Define FOL with an example.
13. Difference between propositional and FOL logic.
14. Define forward chaining and backward chaining.
15. Define Horn clause.
16. Define Canonical horn clause.
17. Write notes on long term and short term memory.
18. Name any 3 frame languages.
19. Write in short about iterative deepening..
20. Is minimax depth fist search or Breadth first search.

PART – B

1. Issues in knowledge representation
2. State Representation of facts in predicate logic.
3. How will you represent facts in propositional logic with an example?
4. Explain Resolution in brief with an example.
5. Write algorithm for propositional resolution and Unification algorithm.
6. Explain in detail about forward and backward chaining with suitable example.
7. Explain steps involved in Matching.
8. Explain the different logics used for knowledge representation.
9. How will you represent facts in Proportional logic with an example.
10. Explain resolution in brief with an example.
11. Explain in detail about minimax procedure.
12. Explain the effect of Alpha beta cut off over minimax.
13. How would the minimax procedure have to be modified to be used by a program playing 3 or 4 persons instead of 2 persons.

.