UNIT-1
PART-A

1. **What is AI technique?**

   Artificial Intelligence (AI) is the study of how to make computers do things which, at the moment, people do better.

   It is a method that exploits knowledge that should be represented in such a way that:

   - The knowledge captures generalizations.
   - It can be understood by people who must provide it.
   - It can be easily modified to correct errors and to reflect changes.
   - It can be used in situations even if it is not accurate and complete.
   - It helps to narrow the range of possibilities that is has to be considered.

2. **What are the steps to be considered to solve a problem?**

   To solve a particular problem, the following steps are taken into account,

   - Define the problem precisely.
   - Analyze the problem.
   - Isolate and represent the task knowledge that is necessary to solve a problem.
   - Choose the best problem-solving techniques and apply it to a particular problem.

3. **How state space representation is is related to the problem solving?**

   The structure of state space representation corresponds to the structure of problem solving in two important ways:

   - It allows for a formal definition of a problem as the need to convert some given situation into some desired situation using a set of permissible operations.
   - It permits to define the process of solving a particular problem as a combination of known techniques. Search is a very important process in the solution of hard problems for which no more direct techniques are available.

4. **What is production systems?**

   A production system consists of:

   - A set of rules that describes the operations and applicability of the rule.
   - One or more knowledge/databases that contains the informations appropriate for the particular task.
   - A control strategy that specifies the order in which the rules will be compared to the database and a way of resolving the conflicts that arise when several rules match at once.
   - A rule applier.

5. **Write the Breadth-First Search algorithm.**

   1. Create a variable called NODE-LIST and set it to the initial state.
   2. Until the goal state is found or NODE-LIST is empty:
      a) Remove the first element from the NODE-LIST and call it E. If NODE-LIST was empty, quit.
      b) For each way that each rule can match the state described in E do,
         i. Apply the rule to generate a new state.
         ii. If the new state is a goal state, quit and return this state.
         iii. Otherwise, add the new state to the end of NODE-LIST.

6. **What are the advantages of Breadth-First Search algorithm?**
    - Breadth-first search will not get trapped by following a single, unfruitful path for a very long time.
    - If there is a solution, then breadth-first search is guaranteed to find it.
    - Longer paths are never explored until all shorter ones have already been examined.
7. **Write the Depth-First Search algorithm.**
    1. If the initial state is a goal state, quit and return success.
    2. Otherwise, do the following until success or failure is signaled.
        a) Generate the successor, E, of the initial state. If there are no more successors, signal failure.
        b) Call Depth-first search with E as the initial state.
        c) If success is returned, signal success. Otherwise continue in this loop.
8. **What are the advantages of Depth-First Search algorithm?**
    - Depth-first search requires less memory since only the nodes on the current path are stored.
    - It may find the solution without examining much of the search space at all. The process can be stopped once the solution is found.
9. **What are the different dimensions to be analyzed for a problem?**
    - Is the problem decomposable into a set of independent smaller or easier sub problems?
    - Can solution steps be ignored or at least undone if they prove unwise?
    - Is the problem's universe predictable?
    - Is the good solution to the problem obvious without comparison to all other possible solutions?
    - Is the desired solution a state of the world or a path to the state?
    - Is a large amount of knowledge absolutely required to solve a problem, or is knowledge important only to constraint the search?
    - Will the solution of the problem requires interaction between the computer and a person?
10. **What are the three classes of problem?**
    - **Ignorable**, in which solution steps can be ignored.
    - **Recoverable**, in which solution steps can be undone.
    - **Irrecoverable**, in which solution steps cannot be undone.
11. **How is production system classified?**
    - **Monotonic production system**- It is a production system in which the application of a rule never prevents the later application of another rule that could also have been applied at the time the first rule was selected.
    - **Partially commutative production system**- It is a production system with the property that if the application of a particular sequence of rules transforms state x and state y, then any permutations of those rules that is allowable also transforms state x into state y.
    - **Commutative production system**- It is a production system that is both monotonic and partially commutative.
12. **What are the important issues of general-purpose search techniques?**
    - The direction in which to conduct the search (forward versus backward reasoning).
    - How to select applicable rules (matching).

- How to represent each node in the search process (the knowledge representation problem and the frame problem).

13. What are the problems faced by hill-climbing search?

Hill-climbing often get stuck for the following reasons :

- Local maxima – A local maxima is a peak that is higher than each of its neighboring states, but lower than the local maximum. Hill climbing algorithm that reach the vicinity of a local maximum will be drawn upwards towards the peak, but will then be stuck with nowhere else to go.
- Ridges – Ridges result in a sequence of local maxima that is very difficult for greedy algorithms to navigate.
- Plateau – a plateau is an area of state space landscape where the evaluation function is flat. A hill-climbing search might be unable to find its way off the plateau.

14. How can we avoid ridge and plateau in hill climbing? (NOV/DEC 2012)

Ridges result in sequence of local maxima that is very difficult for greedy algorithm to navigate. A plateau is an area to the state space landscape where the evaluation function is flat. It can be a flat local maximum, from which no uphill exit exists, or a shoulder, from which it is possible to make progress. In case of plateau a sideways move is allowed in hope that the plateau is really a shoulder. If a sideways move is always allowed an infinite loop might occur. So a limit is placed on the number of consecutive sideways moves allowed.

15. List the criteria to measure the performance of search strategies. (MAY/JUNE 2014)

The criteria to measure the performance of search strategies are:

- Completeness: is the algorithm guaranteed to find a solution when there is one?
- Optimality: does the strategy find the optimal solution?
- Time complexity: how long does it take to find a solution?
- Space complexity: how much memory is needed to perform the search?

16. Define heuristics. Why are heuristics crucial for the efficient design of an expert system?

Heuristics is the study of the methods and rules of discovery and invention. In state space search, heuristics define the rules for choosing branches in a state space that are most likely to lead to an acceptable solution. There are two cases in AI searches when heuristics are needed:

- The problem has no exact solution. For example, in medical diagnosis doctors use heuristics to choose the most likely diagnoses given a set of symptoms. (medical expert systems)
- The problem has an exact solution but is too complex to allow for a brute force solution.

Key Point: Heuristics are fallible. Because they rely on limited information, they may lead to a suboptimal solution or to a dead end.

17. State the significance of using heuristic functions. (NOV/DEC 2011)

A heuristic function is used to estimate the cost of cheapest path from node n to a goal node.

18. What is AND-OR graphs?

- AND-OR graph (or tree) are useful for representing the solution of problems that can be solved by decomposing them into a smaller problems.
- One AND arc may point to any number of successor nodes.
- Several arcs may emerge from a single node, indicating a variety of ways in which the problem can be solved.

19. What is Constraint satisfaction?

Constraint satisfaction is a search procedure that operates in a space of constraint sets. The initial state contains the constraints that are originally given in the problem description. A goal state is any state that has been constrained "enough", where "enough" must be defined for each problem.

20. What are the steps involved in Constraint satisfaction?

- Constraints are discovered and propogated as far as possible throughout the system.
- If the union of constraints defines a solution, then quit and report the solution.

- If the union of constraints defines a contradiction, then return failure.
- If neither of the above occurs, then it is necessary to make a guess at something in order to proceed.

21. What are the significant considerations of Constraint satisfaction?

- Constraints are propagated by using rules that correspond to the properties of arithmetic.
- A value is guessed for some letter whose values are not determined.

22. What are the different kinds of constraints?

- **Simple Constraints-** They list possible values for a single object.
- **Complex Constraints-** They describe relationships between or among objects.

23. Define a CSP.

A constraint satisfaction problem or CSP is defined as a set of variables, X_1, X_2,……,X_n and a set of constraints C_1, C_2,……,C_m. Each variable X_i has a nonempty domain D_i of possible values. Each constraint C_i involves some subset of the variables and specifies the allowable combinations of values for that subset. A state of the problem is defined by an assignment of values to some or all of the variables. An assignment that does not violate any constraints is called a consistent or legal assignment and a solution to a CSP is a complete assignment that satisfies all the constraints.

24. What is means-end analysis?

The means-end analysis process centers around the detection of differences between the current state and the goal state. Once the difference is isolated, an operator that reduces the differences must be found but cannot be applied to the current state.

25. What is operator subgoaling?

The kind of backward chaining in which operators are selected and then subgoals are setup to establish the preconditions of the operators is called operator subgoaling.

PART-B

1. **(i) Describe a state space in which iterative deepening search performs much worse than depth-first search.**

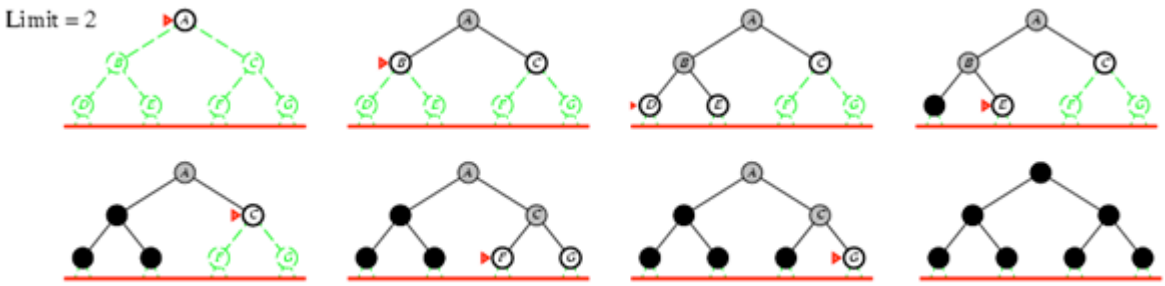
To avoid the infinite depth problem of DFS, we can decide to only search until depth L, i.e. we don't expand beyond depth L. This is called Depth-Limited Search. Suppose if the solution is deeper than L, then increase L iteratively. This is known as Iterative Deepening Search. This iterative deepening search inherits the memory advantage of Depth-First search.

Iterative deepening looks inefficient because so many states are expanded multiple times. In practice this is not that bad, because by far most of the nodes are at the bottom level.

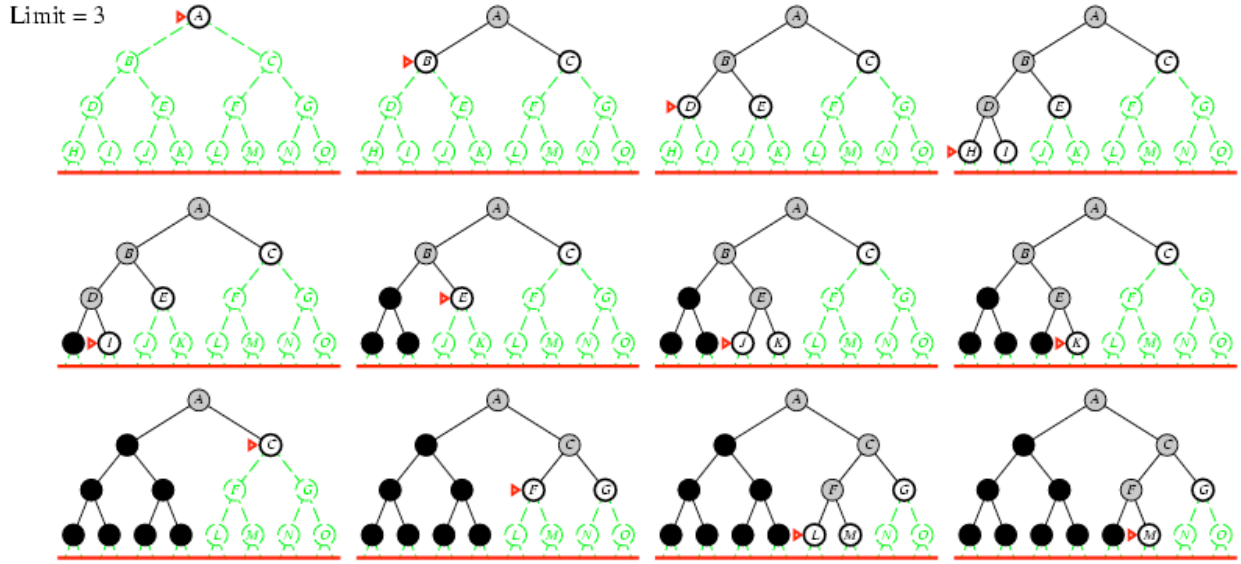
- For a branching factor b of 2, this might double the search time.
- For a branching factor b of 10, this might add 10% to the search time.

Iterative deepening search *L*=1

Iterative deepening search *L*=2



Iterative deepening search *L*=3



- Number of nodes generated in a depth-limited search to depth *d* with branching factor *b*:
  - $N_{DLS} = b^0 + b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d$
- Number of nodes generated in an iterative deepening search to depth *d* with branching factor *b*:
  - $N_{IDS} = (d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d =$ $O(b^d) \neq O(b^{d+1})$
- For *b* = *10*, *d* = *5*,
  - $N_{DLS} = 1 + 10 + 100 + 1{,}000 + 10{,}000 + 100{,}000 = 111{,}111$
  - $N_{IDS} = 6 + 50 + 400 + 3{,}000 + 20{,}000 + 100{,}000 = 123{,}450$
- NBFS = ... = 1,111,100

Performance Analysis

Completeness - Yes

Time Complexity- $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

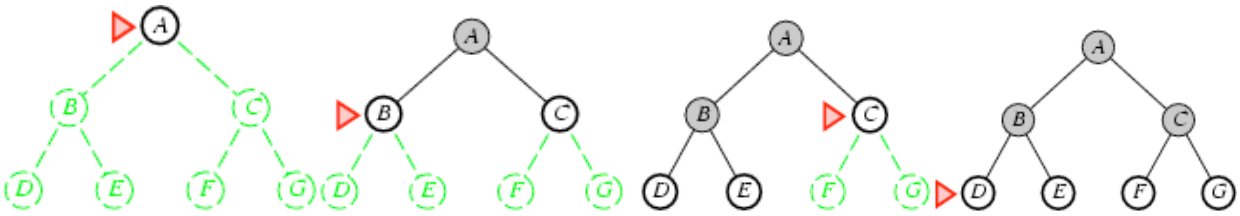
Space Complexity - $O(bd)$

Optimality - Yes, if step cost = 1 or increasing function of depth.

(ii) Prove that the breadth first search is a special case of uniform cost search.

Breadth first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a first-in-first-out (FIFO) queue, i.e., new successors go at end of the queue.



Performance Analysis

Completeness - Yes it always reaches goal (if *b* is finite)

Time Complexity - $1+b+b^2+b^3+\ldots+b^d + (b^{d+1}-b)) = O(b^{d+1})$

- (this is the number of nodes we generate)

Space Complexity - $O(b^{d+1})$ (keeps every node in memory,

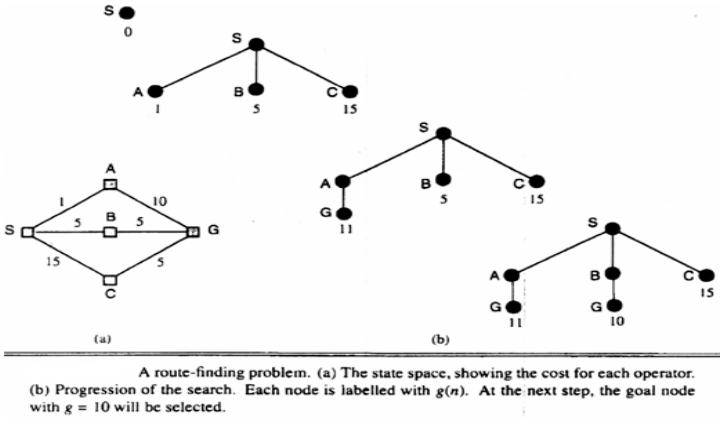
- either in fringe or on a path to fringe).

Optimality - Yes (if we guarantee that deeper solutions are less optimal, e.g. step-cost=1).

Space is the bigger problem (more than time)

Uniform-cost Search

Breadth-first is only optimal if step costs increasing with depth (e.g. constant). Can we guarantee optimality for any step cost? Uniform-cost Search: Expand node with smallest path cost g(n).



A route-finding problem. (a) The state space, showing the cost for each operator. (b) Progression of the search. Each node is labelled with $g(n)$. At the next step, the goal node with $g = 10$ will be selected.

Implementation: *fringe* = queue ordered by path cost

Uniform cost search is equivalent to breadth-first if all step costs all equal.

Performance analysis

Completeness - Yes, if step cost $\geq \varepsilon$ (otherwise it can get stuck in infinite loops)

Time complexity - nodes with *path cost* $\leq$ cost of optimal solution.

Space Complexity - nodes on paths with path cost $\leq$ cost of optimal solution.

Optimality - Yes, for any step cost.

2. **Explain the Control strategies in detail.**

Control Strategy decides which rule to apply next during the process of searching for a solution to a problem.

- Requirements for a good Control Strategy

It should cause motion

In water jug problem, if we apply a simple control strategy of starting each time from the top of rule list and choose the first applicable one, then we will never move towards solution.

It should explore the solution space in a systematic manner

If we choose another control strategy, let us say, choose a rule randomly from the applicable rules then definitely it causes motion and eventually will lead to a solution. But one may arrive to same state several times. This is because control strategy is not systematic.

Systematic Control Strategies (Blind searches)

i) Breadth First Search

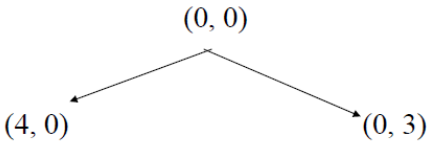
Let us discuss these strategies using water jug problem. These may be applied to any search problem.

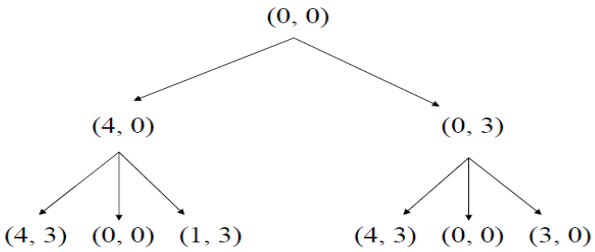
•Construct a tree with the initial state as its root.

•Generate all the offspring of the root by applying each of the applicable rules to the initial state.

•Now for each leaf node, generate all its successors by applying all the rules that are appropriate

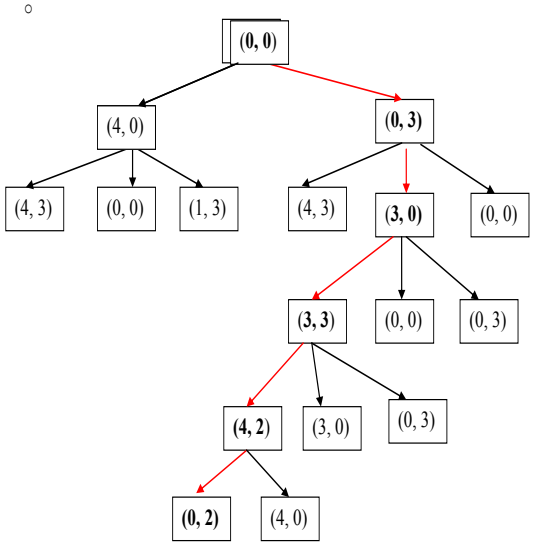
•Continue this process until some rule produces a goal state.





Algorithm:

1. Create a variable called NODE-LIST and set it to initial state
2. Until a goal state is found or NODE-LIST is empty do
   a. Remove the first element from NODE-LIST and call it E. If NODE-LIST was empty, quit
   b. For each way that each rule can match the state described in E do:
      i. Apply the rule to generate a new state
      ii. If the new state is a goal state, quit and return this state
      iii. Otherwise, add the new state to the end of NODE-LIST



Advantages of BFS:

- BFS will not get trapped exploring a blind alley. This contrasts with DFS, which may follow a single unfruitful path for a very long time, perhaps forever, before the path actually terminates in a state that has no successors.
- If there is a solution, BFS is guaranteed to find it.

- If there are multiple solutions, then a minimal solution will be found.

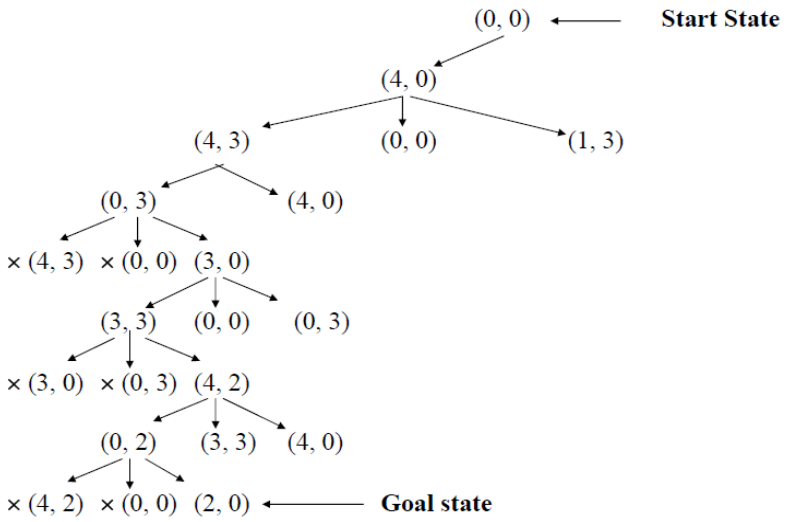
ii) Depth First Search:

Algorithm:

1. If the initial state is a goal state, quit and return success
2. Otherwise, do the following until success or failure is signalled:
    a. Generate a successor, E, of initial state. If there are no more successors, signal failure.
    b. Call Depth-First Search, with E as the initial state
    c. If success is returned, signal success. Otherwise continue in this loop.

Advantages of Depth-First Search:

- DFS requires less memory since only the nodes on the current path are stored.
- By chance, DFS may find a solution without examining much of the search space at all.



3. **Explain how different problem characteristics are analyzed in detail.**

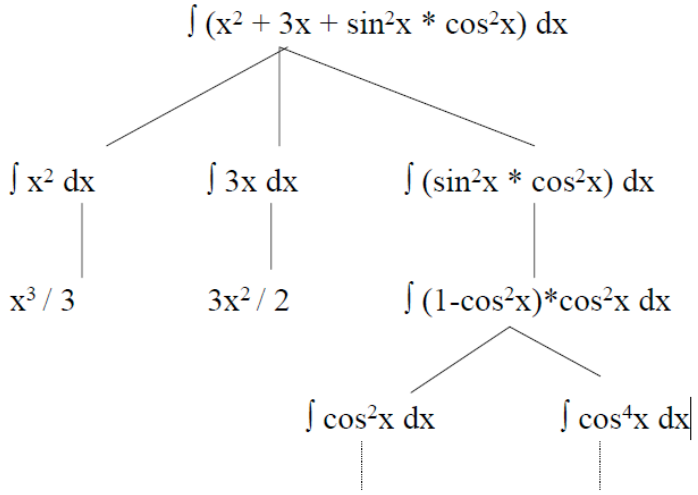
In order to choose the most appropriate method(s) for a particular problem, must analyse the problem along several dimensions.

1. Is the problem decomposable into a set of independent smaller sub problems?

Example: Decomposable Problem

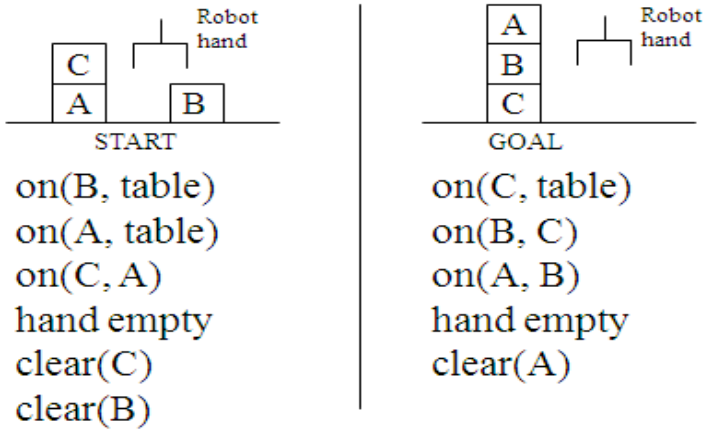
Decomposable problems can be solved by the divide-and-conquer technique. In divide and conquer technique, divide the problem in to sub-problems, find the solutions for the sub-problems. Finally, integrate the solutions for the sub-problems, we will get the solution for the original problem. Suppose we want to solve the problem of computing the integral of the following expression

$\int(x2+ 3x + \sin2x * \cos2x)\, dx$



Example: Non-Decomposable problem

There are non-decomposable problems. For example, Block world problem is non-decomposable.



2. Can Solution Steps be ignored or atleast undone if they prove to be unwise?

In real life, there are three important types of problems:

- Ignorable (theorem proving)
- Recoverable (8-puzzle)
- Irrecoverable (Chess)

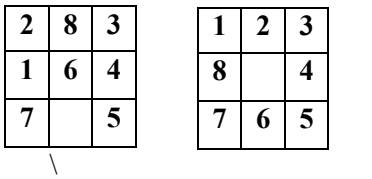
Ignorable (theorem proving)

Suppose we have proved some lemma in order to prove a theorem and eventually realized that lemma is no help at all, then ignore it and prove another lemma.

- It can be solved by using simple control strategy?

Recoverable (8-puzzle)

Objective of 8 puzzle game is to rearrange a given initial configuration of eight numbered tiles on 3 X 3 board (one place is empty) into a given final configuration (goal state). Rearrangement is done by sliding one of the tiles into empty square.



Initial state **Goal state**

- Solved by backtracking

Irrecoverable (Chess)

- A stupid move cannot be undone.
- Can be solved by planning process.

3. Is the universe predictable?

There are two types of problems. Certain outcome and uncertain outcome.

Certain outcome (8-puzzle problem)

Able to plan the entire sequence of moves.

Uncertain outcome (Bridge game)

It is not possible to plan the entire sequence of actions. We do not know exactly where all the cards are or what the other players will do on their turns. To overcome this, investigate several plans and use probabilities of the various outcomes to choose a plan that has the highest estimated probability of leading to a good score on the hand.

4. Is a good solution absolute or relative?

There are two types of problems

- Any path problem
- Best path problem

Any path problem

- Any path problems can often be solved in a reasonable amount of time by using heuristics that suggest good paths to explore.
- Consider a problem of answering questions based on the database of simple facts.

1. Marcus was a man.
2. Marcus was a Pompeian.
3. Marcus was born in 40AD.
4. All men are mortal.
5. All Pompeian died in 79 AD.
6. No mortal lives longer than 150 years.
7. It is now 1991 AD.

Suppose we ask a question ,**"Is Marcus is alive?".** From the facts given we can easily derive an answer to the question.

1. Marcus was a man.	**1**
4. All men are mortal.	**4**
8. Marcus was a mortal.	**1, 4**
3. Marcus was born in 40AD.	**3**
7. It is now 1991 AD.	**7**
9. Marcus age is 1951 years.	**3, 7**
6. No mortal lives longer than 150 years.	**6**
10. Marcus was dead	**6.8.9**

OR

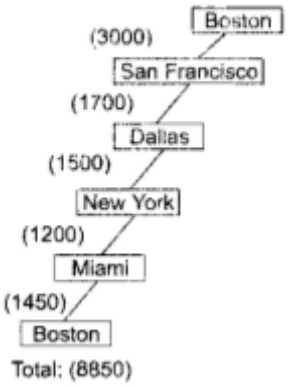
7. It is now 1991 AD.	**7**
5. All Pompeian died in 79 AD.	**5**
11. All Pompeian are dead now.	**7, 5**
2. Marcus was a Pompeian.	**2**
10. Marcus was dead	**11, 2**

Either of two reasoning paths will lead to the answer. We need to find only the answer to the question. No need to go back and see if some other path might also lead to a solution.

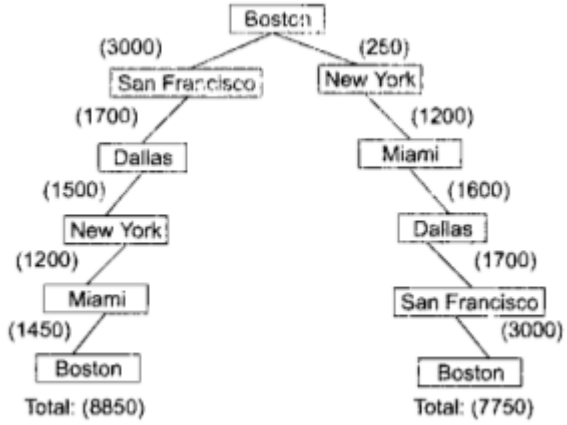
Best path problem

In travelling salesman problem, our goal is to find the shortest route that visits each city exactly once.

	Boston	New York	Miami	Dallas	S.F.
Boston		250	1450	1700	3000
New York	250		1200	1500	2900
Miami	1450	1200		1600	3300
Dallas	1700	1500	1600		1700
S.F.	3000	2900	3300	1700	



This path is not a best solution for the salesman problem.



Best path problems are computationally harder than any path problem.

5. Is the solution a state or path?

State

Finding a consistent interpretation for the sentence "*The bank president ate a dish of pasta salad with the fork*". We need to find the interpretation but not the record of the processing.

Path

In water jug problem, it is not sufficient to report that the problem is solved and the goal is reached, it is (2,0). For this problem, we need the path from the initial state to the goal state.

6. What is the role of knowledge?

For eg, Chess playing

Knowledge about the legal moves of the problem is needed. Without the knowledge, we cannot able to solve the problem.

7. Does the task require Interaction with a person?

Solitary in which the computer is given a problem description and produces an answer with no intermediate communication and no demand for an explanation of the reasoning process.

Conversational in which there is intermediate communication between a person and the computer, either to provide additional assistance to the computer or to provide additional information to the user, or both.

4. **Explain production systems and discuss the major issues in the design of search programs.**

A production system consists of:

- A set of rules, each consisting of a left side that determines the applicability of the rule and a right side that describes the operation to be performed if that rule is applied.
- One or more knowledge/databases that contain whatever information is appropriate for the particular task. Some parts of the database may be permanent, while other parts of it may pertain only to the solution of the current problem.
- A control strategy that specifies the order in which the rules will be compared to the database and a way of resolving the conflicts that arise when several rules match at once.
- A rule applier

Production System is a formation for structuring AI programs which facilitates describing search process.

- It consists of
    - Initial or start state of the problem
    - Final or goal state of the problem
    - It consists of one or more databases containing information appropriate for the particular task.
    - The information in databases may be structured using knowledge representation schemes.

Production system characteristics:

1. Can production systems, like problems, be described by a set of characteristics that shed some light on how they can easily be implemented?
2. If so, what relationships are there between problem types and the types of production systems best suited to solving the problems?

- Classes of Production systems:
    - **Monotonic Production System:**
      Production system in which the application of a rule never prevents the later application of another rule that could also have been applied at the time the first rule was applied.
      i.e., rules are independent.
      Non-Monotonic Production system:
    - **Partially commutative Production system:**
      A partially commutative production system has a property that if the application of a particular sequence of rules transform state x into state y, then any permutation of those rules that is allowable, also transforms state x into state y.
    - **Commutative Production system:**
      A Commutative production system is a production system that is both monotonic and partially commutative.

Partially Commutative, Monotonic:

- These production systems are useful for solving ignorable problems.
- Example: Theorem Proving
- They can be implemented without the ability to backtrack to previous states when it is discovered that an incorrect path has been followed.
- This often results in a considerable increase in efficiency, particularly because since the database will never have to be restored. It is not necessary to keep track of where in the search process every change was made.
- They are good for problems where things do not change; new things get created.

Non Monotonic, Partially Commutative:

- Useful for problems in which changes occur but can be reversed and in which order of operations is not critical.
- Example: Robot Navigation, 8-puzzle, blocks world

- Suppose the robot has the following ops: go North (N), go East (E), go South (S), go West (W). To reach its goal, it does not matter whether the robot executes the N-N-E or N-E-N.

Not partially Commutative:

- Problems in which irreversible change occurs
- Example: chemical synthesis
- The ops can be: Add chemical x to the pot, Change the temperature to t degrees.
- These ops may cause irreversible changes to the potion being brewed.
- The order in which they are performed can be very important in determining the final output.
- **(x+y) +z is not the same as (z+y) +x**
- Non-partially commutative production systems are less likely to produce the same node many times in search process.
- When dealing with ones that describe irreversible processes, it is partially important to make correct decisions the first time, although if the universe is predictable, planning can be used to make that less important.

	Monotonic	NonMonotonic
Partially Commutative	Theorem proving	Robot Navigation
Not Partially Commutative	Chemical Synthesis	Bridge

Issues in the design of search programs

- The direction in which to conduct the search (forward versus backward reasoning). Search forward through the state space from the start sate to goal state or search backward from the goal.
- How to select applicable rules (Matching). Production system typically spend most of their time looking for rules to apply, so it is critical to have efficient procedures for matching rules against states.
- How to represent each node of the search process (knowledge representation problem). For problems like chess, a node can be fully represented by a simple array. For complex problem solving, it is inefficient and/or impossible to represent all the facts in the world and to determine all of the side effects an action may have.

5. **What is heuristic search technique? Explain Hill climbing in detail.**

Heuristic Search (informed search)

A Heuristic is a function that, when applied to a state, returns a number that is an estimate of the merit of the state, with respect to the goal. In other words, the heuristic tells us approximately how far the state is from the goal state.

Heuristics might underestimate or overestimate the merit of a state. But for reasons which we will see, heuristics that only underestimate are very desirable, and are called admissible.

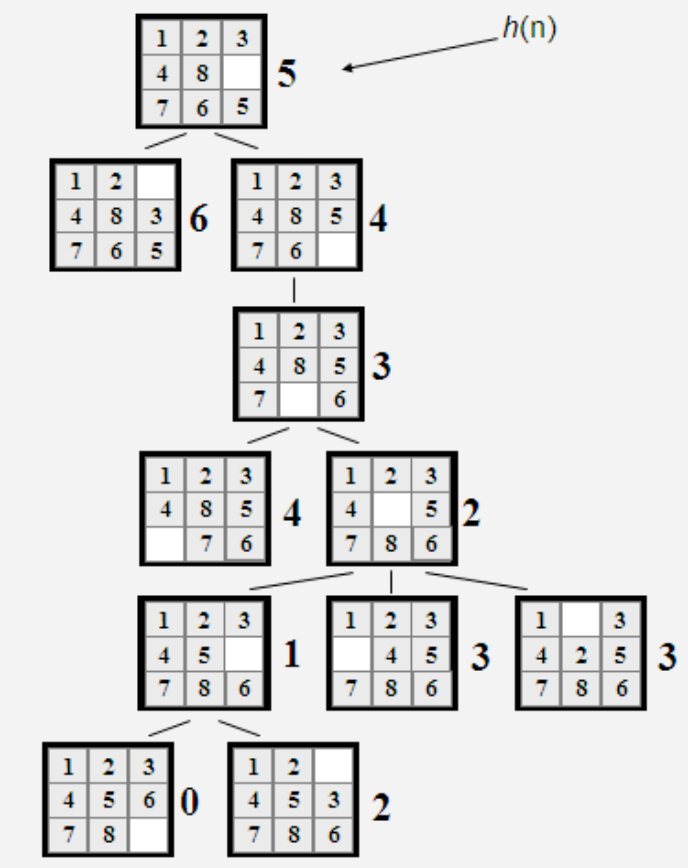
i) Simple Hill Climbing

Algorithm

1. Evaluate the initial state.

2. Loop until a solution is found or there are no new operators left to be applied:
   - Select and apply a new operator
   - Evaluate the new state:
     goal → quit
     better than current state → new current state

Example: 8 puzzle problem
Here, h(n) = the number of **misplaced tiles** (not including the blank), the Manhattan Distance heuristic helps us quickly find a solution to the 8-puzzle.



Advantages of Hill Climbing

- Estimates how far away the goal is.
- Is neither optimal nor complete.
- Can be very fast.

ii) Steepest-Ascent Hill Climbing (Gradient Search):

The steepest ascent hill climbing technique considers all the moves from the current state. It selects the best one as the next state.

Algorithm

1. Evaluate the initial state.
2. Loop until a solution is found or a complete iteration produces no change to current state:
   - SUCC = a state such that any possible successor of the current state will be better than SUCC (the worst state).
   - For each operator that applies to the current state, evaluate the new state:
     goal → quit
     better than SUCC → set SUCC to this state
   - SUCC is better than the current state → set the current state to SUCC.

Disadvantages

- **Local maximum**

A state that is better than all of its neighbours, but not better than some other states far away.

- **Plateau**

A flat area of the search space in which all neighbouring states have the same value.

- **Ridge**

The orientation of the high region, compared to the set of available moves, makes it impossible to climb up. However, two moves executed serially may increase the height.

There are some ways of dealing with these problems.

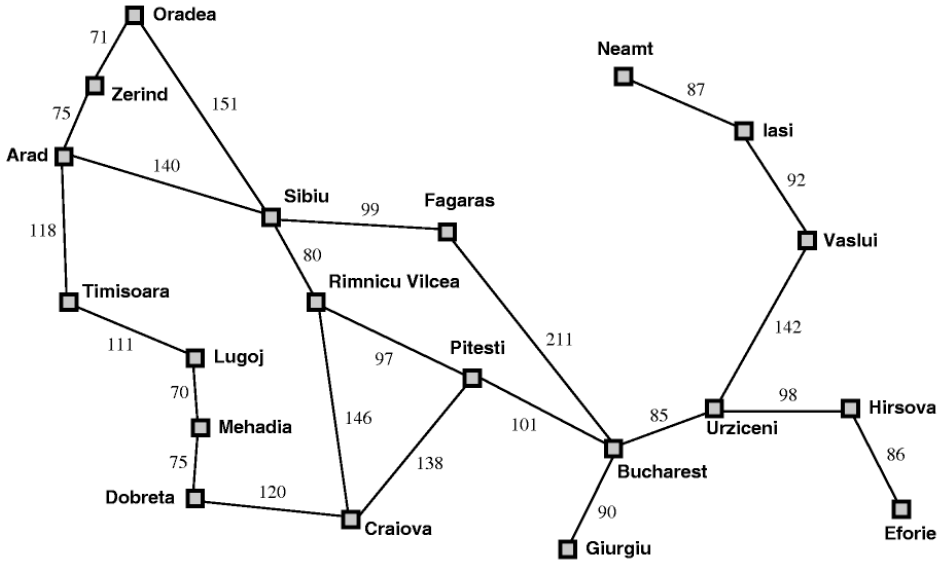
- Backtrack to some earlier node and try going in a different direction. This is a good way of dealing with local maxima.
- Make a big jump to try to get in a new section of the search space. This is particularly a good way of dealing with plateaus.
- Apply two or more rules before doing a test. This corresponds to moving in several directions at once. This is particularly a good strategy for dealing with ridges.

6. **Explain Best-first search algorithm in detail.**

Best first search combines the advantages of Breadth-First and Depth-First searches.

  - DFS: follows a single path, don't need to generate all competing paths.
  - BFS: doesn't get caught in loops or dead-end-paths.

- Best First Search: explore the most promising path seen so far. Nodes are ordered and expanded using evaluation function. The best evaluation function is expanded first.
- Two types of evaluation function
  - **Greedy Best First Search**
  - **A* search**



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Best First Search

Greedy best first search minimize the estimated cost to reach the goal. It always expand the node that appears to be closed to the goal.

Evaluation function

f(n)=h(n)

- *h(n) = estimated cost of the cheapest path from the state at node n to a goal state*

Algorithm

1. Start with OPEN containing just the initial state.
2. Until a goal is found or there are no nodes left on OPEN do

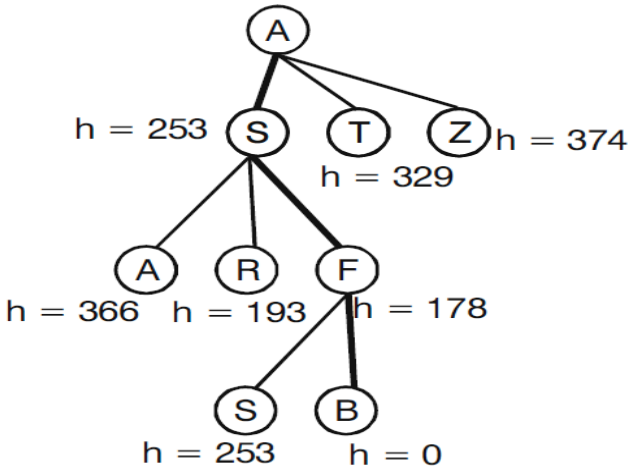
(a) Pick the best node on OPEN

(b) Generate its successors

(c) For each successor do

(i) If it has not been generated before, evaluate it, add it to OPEN, and record its parent.

(ii) If it has been generated before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already have.



A-S-F-B→ 140+99+211=450

Performance Analysis

- Time and space complexity – $O(b^m)$
- Optimality – no
- Completeness - no

A* search Algorithm

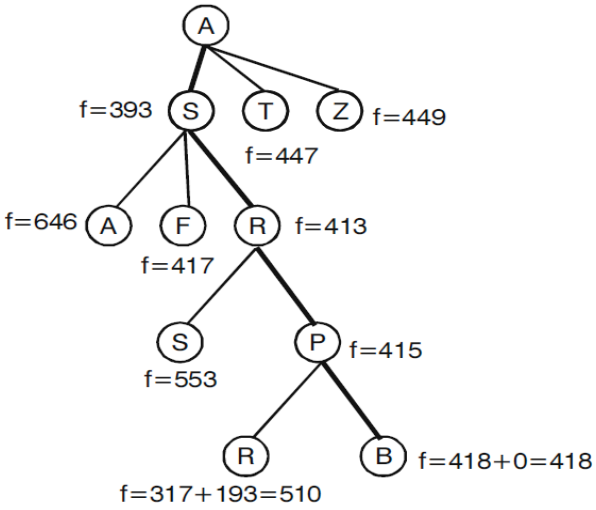
Evaluation function

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

- f(n) = cost of the cheapest solution through n
- g(n) = actual path cost from the start node to node n

Algorithm

1. Create a priority queue of search nodes (initially the start state). Priority is determined by the function f)
2. While queue not empty and goal not found:
   (a) Get best state x from the queue.
   (b) If x is not goal state:
   (i) generate all possible children of x (and save path information with each node).
   (ii) Apply f to each new node and add to queue.
   (iii) Remove duplicates from queue (using f to pick the best).



A-S-R-P-B→ 140+80+97+101=418

Performance Analysis

- Time complexity – depends on heuristic function and admissible heuristic value
- space complexity – $O(b^m)$
- Optimality – yes (locally finite graphs)
- Completeness – yes (locally finite graphs)

7. **Write algorithm for the following:**

i. Generate-and-test

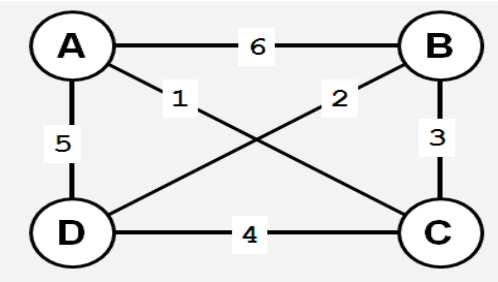
The generate and test is the simplest form of all heuristic search methods

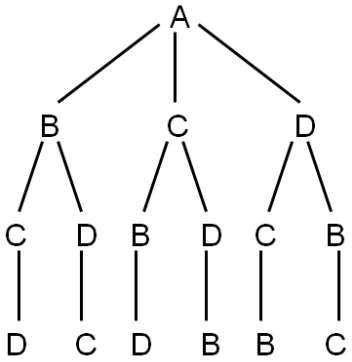
Algorithm

1. Generate a possible solution. For some problems, this means generating a particular point in the problem space. For others, it means generating a path from a start state.
2. Test to see if this is actually a solution by comparing the chosen point or the endpoint of the chosen path to the set of acceptable goal states.
3. If a solution has been found, quit. Otherwise, return to step 1.

Example - Traveling Salesman Problem (TSP)

- Traveler needs to visit *n* cities.
- Know the distance between each pair of cities.
- Want to know the shortest route that visits all the cities once.





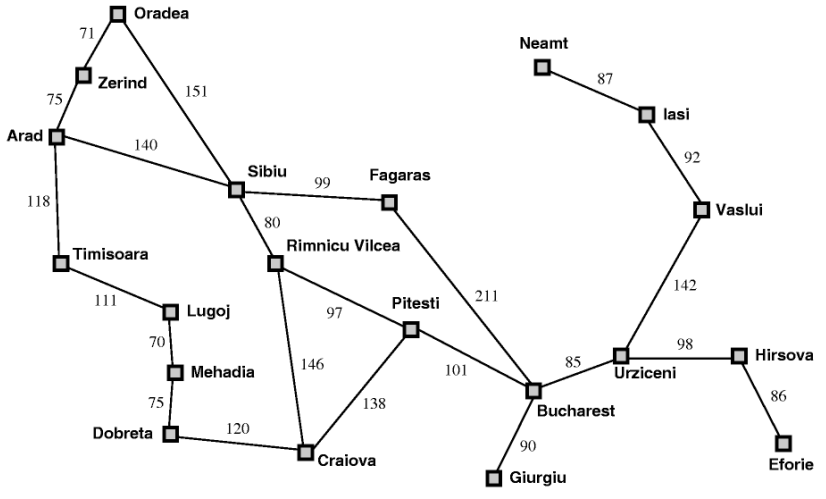
- TSP - generation of possible solutions is done in lexicographical order of cities:
  1. A - B - C - D
  2. A - B - D - C
  3. A - C - B - D
  4. A - C - D - B
  5. A - D - C - B
  6. A - D - B - C
- *n=80* will take millions of years to solve exhaustively!

ii. A* Algorithm

Evaluation function

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

- f(n) = cost of the cheapest solution through n
- g(n) = actual path cost from the start node to node n

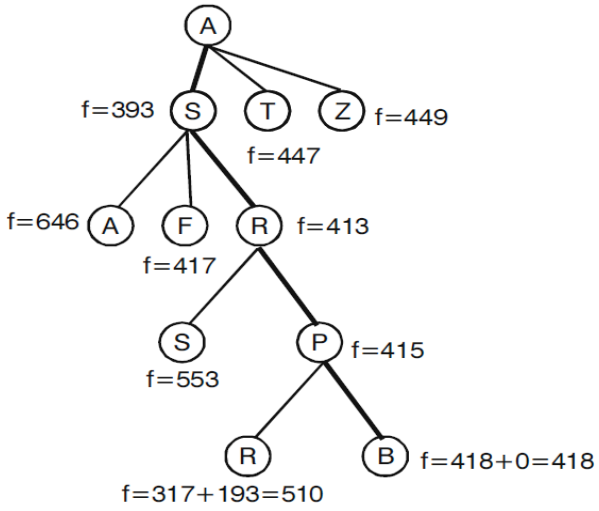


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Algorithm

1. Create a priority queue of search nodes (initially the start state). Priority is determined by the function f)
2. While queue not empty and goal not found:
   (a) Get best state x from the queue.
   (b) If x is not goal state:

(i) generate all possible children of x (and save path information with each node).
(ii) Apply f to each new node and add to queue.
(iii) Remove duplicates from queue (using f to pick the best).



A-S-R-P-B→ 140+80+97+101=418

Performance Analysis

- Time complexity – depends on heuristic function and admissible heuristic value
- space complexity – $O(b^m)$
- Optimality – yes (locally finite graphs)
- Completeness – yes (locally finite graphs)

iii. Agenda-driven search

An agenda is a list of tasks a system could perform. Associated with each task there are usually two things: a list of reasons why this task is being proposed and a rating representing the overall weight of evidence suggesting that the task would be useful.

Algorithm

Do until a good state is reached or the agenda is empty

(a) Choose the most promising task from the agenda. Notice that this task can be represented in any desired form. It can be thought of as an explicit statement of what to do next or simply as an indication of the next node to be expanded.

(b) Execute the task by devoting to it the number of resources determined by its importance. The important resources are time and space. Executing the task will generate the different task will probably generate additional tasks. For each task, do the following,

i) See if it is already on the agenda. If so, then see if this same reason for doing it is already on the list of justifications. . If so, ignore the current evidence. If this justification was not already present, add the agenda justification to the list. If the task was not on the agenda, insert it. ii) Compute the new task's rating, combining the evidences from all its justifications. Not all justification have equal weight. It is often usual to associate with each justification a measure of how strong a reason it is. These measures are then combined at this step to produce an overall rating for that task.

- Agenda driven control structure is also useful if some tasks provide negative evidence about the merits of other tasks. This can be represented by justifications with negative weightings.
- Agenda mechanism provides a good way of focusing the attention of complex system in the areas suggested by the greatest number of positive indicators.

iv. Means-end analysis

Means end analysis allows both backward and forward searching. Search process reduces the difference between the current state and the goal state until the required goal is achieved

- Solve major parts of a problem first and then return to smaller problems when assembling the final solution – operator sub-goaling
- Example :
  - GPS was the first AI program to exploit means-ends analysis.
  - STRIPS (A robot Planner)

Procedure

1. Until the goal is reached or no more process are available:
   (a) Describe the current state, the goal state and the differences between the two.
   (b) Use the difference between the current state and goal state, possibly with the description of the current state or goal state, select a promising procedure.
   (c) Use the promising procedure and update current state.
2. If goal is reached then **success** otherwise **failure**.

Household robot domain

- Problem: Move desk with two things on it from one location S to another G. Find a sequence of actions robot performs to complete the given task.
- Operators are: PUSH, CARRY, WALK, PICKUP, PUTDOWN and PLACE given with preconditions and results.

S B_____C G

Start PUSH Goal

S(Start)→ walk(start_desk_loc) → pickup(obj1) → putdown(obj1) → pickup(obj2)→ putdown(obj2) →push(desk, goal_loc) → walk(start_desk_loc) → pickup(obj1) → carry(obj1,goal_loc) → place(obj1,desk) → walk(start_desk_loc) → pickup(obj2) → carry(obj2,goal_loc) → place(obj2,desk) →G(Goal).

Algorithm

1. Compare CURRENT and GOAL. If there are no differences between them then return.

2Otherwise, select the most important difference and reduce it by doing the following until success or failure is signaled.

(a) Select an as yet untried operator O that is applicable to the current difference. If there are no such operators, then signal failure.

(b) Attempt to apply O to CURRENT. Generate descriptions of two states:

O-START- a state in which O's preconditions are specified.

O-RESULT- the state that would result if O were applied in O-START.

(c) If

(FIRST-PART←MEA(CURRENT, O-START))

and

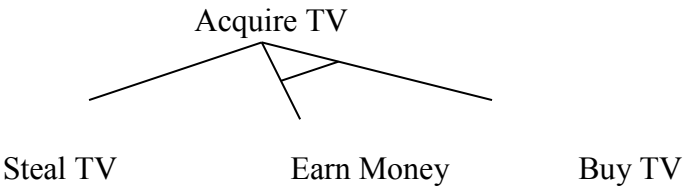
(LAST-PART←MEA(O-RESULT, GOAL))

are successful, then signal success and return the result of concatenating FIRST-PART, O and LAST-PART.

8. **Explain problem-reduction algorithm in detail.**

Problem Reduction:

- Each sub-problem is solved and final solution is obtained by combining solutions of each sub-problem.
- Decomposition generates arcs that we will call AND arc.
- One AND arc may point to any number of successors, all of which must be solved.
- Such structure is called AND–OR graph rather than simply AND graph.

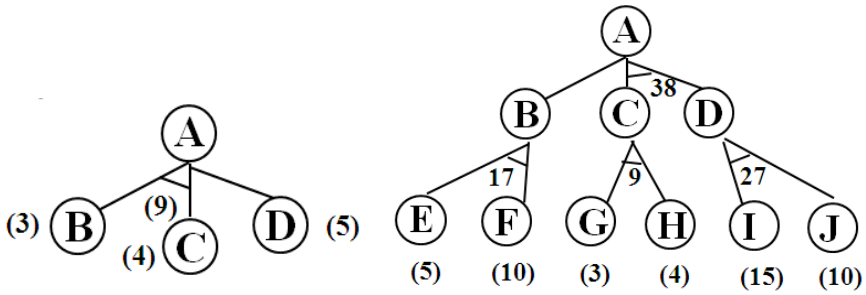


To find a solution in AND–OR graph, we need an algorithm similar to A* with the ability to handle AND arc appropriately.

In search for AND-OR graph, we will also use the value of heuristic function **f** for each node.

i) AND–OR Graph Search:

- Traverse AND-OR graph, starting from the initial node and follow the current best path.
- Accumulate the set of nodes that are on the best path which have not yet been expanded.
- Pick up one of these unexpanded nodes and expand it.
  Add its successors to the graph and compute f (using only h) for each of them.
- Change the f estimate of newly expanded node to reflect the new information provided by its successors.
  Propagate this change backward through the graph to the start.
- Mark the best path which could be different from the current best path.
- Propagation of revised cost in AND-OR graph was not there in A*.



Algorithm: Production reduction

1. Initialize the graph to the starting node.
2. Loop until the starting node is labelled SOLVED or until its cost goes above FUTILITY:

   a) Traverse the graph, starting at the initial node and following the current best path and accumulate the set of nodes that are on the path and have not yet been expanded or labelled as solved.

   b) Pick one of the unexpanded nodes and expand it. If there are no successors, assign FUTILITY as the value of this node. Otherwise, add its successors to the graph and for each of them complete f' (use only h' and ignore g) .if any node is 0,mark that node as SOLVED.

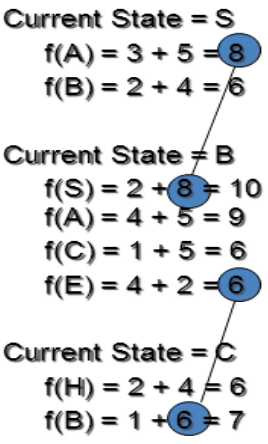
   c) Change the f' estimate of the newly expanded need to reflect the new information provided by its successors. Propagate this change backward through the graph. If any node contains the successor are whose descendants are all solved, label the node itself as SOLVED. At each that us visited while going up the graph, decide which of its successor

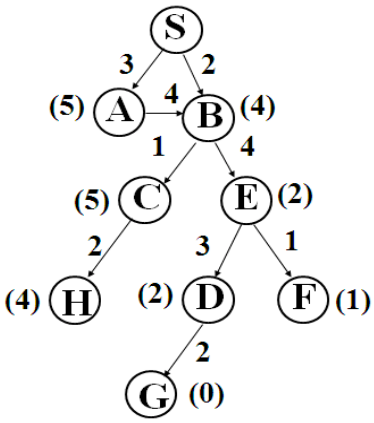
arcs is the most promising and mark it as part of the current best path. This may cause the current best path to change. This prorogation of revised cost estimates back up the tree was not necessary in the best first search algorithm.

ii) AO* algorithm

AO* algorithm uses a single structure GRAPH, representing the part of the search graph that has been explicitly generated so far. Each node in the graph will point both down to its immediate successors and up to its immediate predecessors.

1. Let G be a graph with only starting node INIT.
2. Repeat the followings until INIT is labelled SOLVED or h(INIT) > FUTILITY
   a) Select an unexpanded node from the most promising path from INIT (call it NODE)
   b) Generate successors of NODE. If there are none, set h(NODE) = FUTILITY (i.e., NODE is unsolvable); otherwise for each SUCCESSOR that is not an ancestor of NODE do the following:
      i. Add SUCCESSSOR to G.
      ii. If SUCCESSOR is a terminal node, label it SOLVED and set h(SUCCESSOR) = 0.
      iii. If SUCCESSPR is not a terminal node, compute its h.
   c) Propagate the newly discovered information up the graph by doing the following: let S be set of SOLVED nodes or nodes whose h values have been changed and need to have values propagated back to their parents. Initialize S to Node. Until S is empty repeat the followings:
      i. Remove a node from S and call it CURRENT.
      ii. Compute the cost of each of the arcs emerging from CURRENT. Assign minimum cost of its successors as its h.
      iii. Mark the best path out of CURRENT by marking the arc that had the minimum cost in step ii
      iv. Mark CURRENT as SOLVED if all of the nodes connected to it through new labelled arc have been labelled SOLVED
      v. If CURRENT has been labelled SOLVED or its cost was just changed, propagate its new cost back up through the graph. So add all of the ancestors of CURRENT to S.



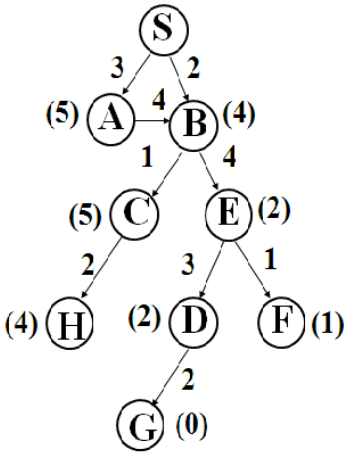


Current State = H
f(C) = 2 + 7 = 9

Current State = C
f(B) = 1 + 6 = 7
f(H) = ∞

Current State = B
f(S) = 2 + 8 = 10
f(A) = 4 + 5 = 9
f(E) = 4 + 2 = 6
f(C) = ∞

Current State = E
f(B) = 4 + 9 = 13
f(D) = 3 + 2 = 5
f(F) = 1 + 1 = 2



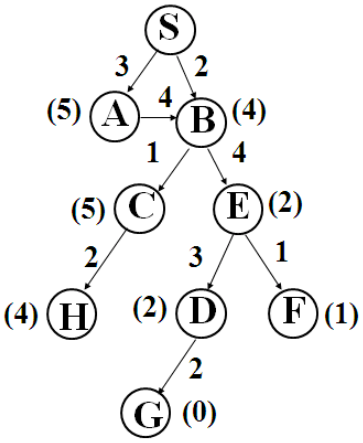
Current State = F
f(E) = 1 + 5 = 6

Current State = E
f(D) = 3 + 2 = 5
f(B) = 4 + 9 = 13
f(F) = ∞

Current State = D
f(G) = 2 + 0 = 2
f(E) = 3 + 13 = 16

Visited Nodes =
S, B, C, H, C, B, E, F, E, D, G

Path = S, B, E, D, G



9. **Explain AO* algorithm with a suitable example. State the limitations in the algorithm.**

AO* algorithm:

AO* algorithm uses a single structure GRAPH, representing the part of the search graph that has been explicitly generated so far. Each node in the graph will point both down to its immediate successors and up to its immediate predecessors.

3. Let G be a graph with only starting node INIT.
4. Repeat the followings until INIT is labelled SOLVED or h(INIT) > FUTILITY
    a) Select an unexpanded node from the most promising path from INIT (call it NODE)
    b) Generate successors of NODE. If there are none, set h(NODE) = FUTILITY (i.e., NODE is unsolvable); otherwise for each SUCCESSOR that is not an ancestor of NODE do the following:
        i. Add SUCCESSSOR to G.
        ii. If SUCCESSOR is a terminal node, label it SOLVED and set h(SUCCESSOR) = 0.

iii. If SUCCESSPR is not a terminal node, compute its h.

c) Propagate the newly discovered information up the graph by doing the following: let S be set of SOLVED nodes or nodes whose h values have been changed and need to have values propagated back to their parents. Initialize S to Node. Until S is empty repeat the followings:

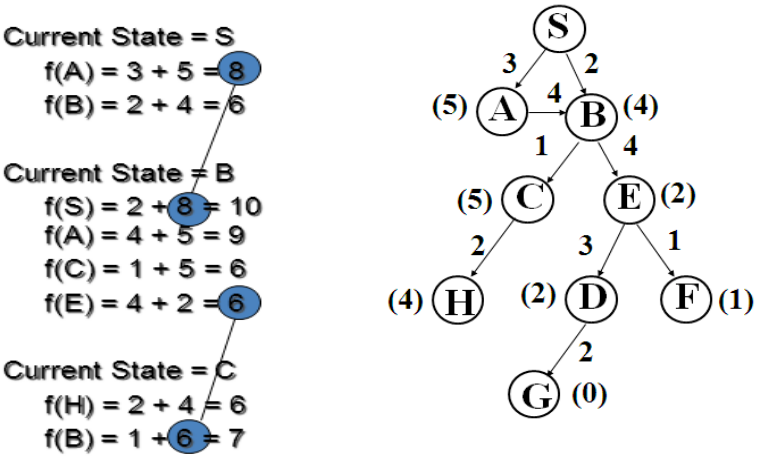
i. Remove a node from S and call it CURRENT.

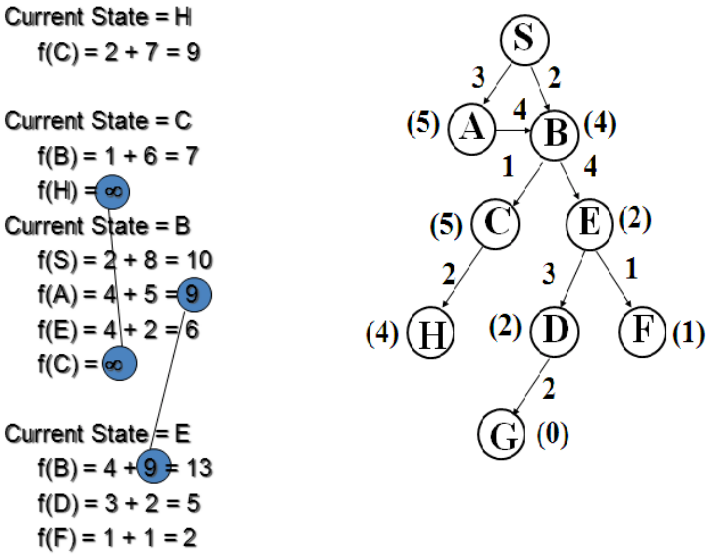
ii. Compute the cost of each of the arcs emerging from CURRENT. Assign minimum cost of its successors as its h.

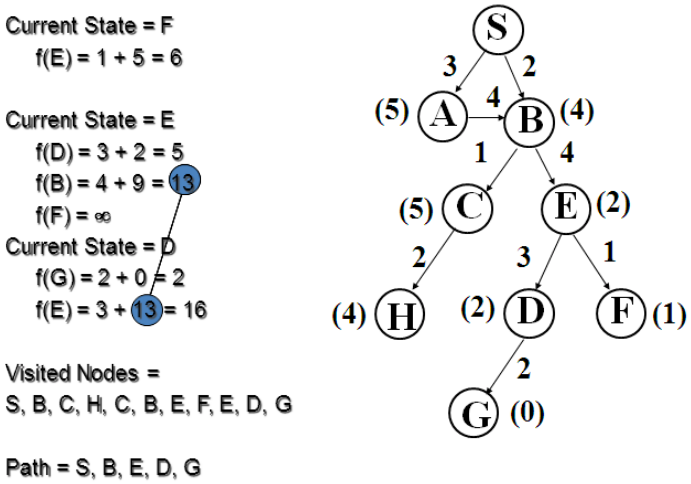
iii. Mark the best path out of CURRENT by marking the arc that had the minimum cost in step ii

iv. Mark CURRENT as SOLVED if all of the nodes connected to it through new labelled arc have been labelled SOLVED

v. If CURRENT has been labelled SOLVED or its cost was just changed, propagate its new cost back up through the graph. So add all of the ancestors of CURRENT to S.







10. **Explain the constraint satisfaction procedure to solve the crypt arithmetic problem.**

C R O S S
+ R O A D S
D A N G E R **(NOV/DEC 2011)**

Constraint satisfaction is a two-step process

- Constraints are discovered and propagated as far as possible throughout the system. Then, if there is still not a solution, search begins.
- A guess about something is made and added as a new constraint. Propagation can then occur with this new constraint, and so forth

Algorithm

1. Propagate available constraints. To do this, first set OPEN to the set of all objects that must have values assigned to them in complete solution. Then do until an inconsistency is detected or until OPEN is empty:
   (a) Select an object OB from OPEN. Strengthen as much as possible the set of constraints that apply to OB.
   (b) If this set is different from the set that was assigned the last time OB was examined or if this is the first time OB has been examined, then add to OPEN all objects that share any constraints with OB.
   (c) Remove OB from OPEN.

2. If the union of the constraints discovered above defines a solution , then quit and report the solution.
3. If the union of the constraints discovered above defines a contradiction, then return failure.
4. If neither of the above occurs, then is necessary to make a guess at something inorder to proceed. To do this, loop until a solution is found or all possible solutions have been eliminated.
   (a) Select an object whose value is not yet determined and select a way of strengthening the constraints on the object.
   (b) Recursively invoke constraint satisfaction with the current set of constraints augmented by the strengthening constraint just selected.

Example

Given : CROSS + ROADS = DANGER
Initial state : C R O S A D N G E C1 C2 C3 C4 = ?
Goal state: The digits to the letter should be assigned in such a manner that the sum is satisfied.

Let S=2

S(2) + S(2) = R(4)

Let D=1

S(2) + D(1) = E(3)

Let A=5

O + A(5) = G

To find the value for O let us apply the digits 6,7,8 or 9, which derives the G value has 1,2,3 or 4 . But these digit values are already assigned to some other letters. So the value of O should be greater than 9, but it is a failure one as per the operator is concerned.

Let A = 6

O + A(6) = G

O>8, (i.e) O=9

O(9) + A(6) = G(5)

C3(1) + R (4) +O(9) = N(4)

Contradiction state, (i.e) the value of N and R are same.

Let S=3

S(3) +S(3)=R(6)

Let D=1

S(3) + D(1)=E(4)

Let O=2

A + O(2)=G

Let A=5

A(5) + O(2)=G(7)

Let R=6

R(6) + O(2)=N(8)

C + R(6)=A(5)

$\therefore C=9$

	C4(0)	C3(0)	C2(0)	C1(0)	
	C(9)	R(6)	O(2)	S(3)	S(3)
	R(6)	O(2)	A(5)	D(1)	S(3)
D(1)	A(5)	N(8)	G(7)	E(4)	R(6)

UNIT-2
PART-A

1. **What is Knowledge representation?**
   Knowledge representation is the field of artificial intelligence (AI) dedicated to representing information about the world in a form that a computer system can utilize to solve complex tasks such as diagnosing a medical condition or having a dialog in a natural language.
2. **What are the levels of knowledge representation?**
   - **Knowledge level,** at which facts such as agent's behaviors and current goals are described.
   - **Symbol level,** at which representation of objects at the knowledge level are defined in terms of symbols that can be manipulated by the programs.
3. **Define facts.**
   Facts are truth in some relevant world and they are represented in some chosen formalism.
4. **What are forward and backward representation mappings?**
   The forward representation mapping maps from facts to representation whereas backward representation mapping maps from representation to facts.
5. **What are the properties of a good system for the representation of knowledge?**
   Representational Adequacy- the ability to represent all of the kinds of knowledge that are needed in that domain.
   Inferential Adequacy- the ability to manipulate the representational structures in such a way as to derive new structures corresponding to new knowledge inferred from old.
   Inferential Efficiency- the ability to incorporate into the knowledge structure additional information that can be used to focus the attention of the inference mechanisms in the most promising directions.
   Acquisitional Efficiency- the ability to acquire new information easily.
6. **What are the issues in knowledge representation?**
   - Are any attributes of objects so basic that they occur in almost every problem domain? If so, What are the attributes?
   - Are there any important relationships that exist among attributes of objects?
   - At what level should knowledge be represented? Is there a good set of primitives into which all knowledge can be broken down? Is it helpful to use such primitives?
   - How should set of objects be represented?
   - Given a large amount of knowledge stored in a database, how can relevant parts be accessed when they are needed?
7. **Define predicate logic.**
   Predicate logic is an extension of propositional logic to formulas involving terms and predicates.
   It is un-decidable but serves as a useful way of representing and manipulating some of the kind of knowledge that an AI system might need.
8. **Represent the following sentence in predicate form "All the children likes sweets". (NOV/DEC 2012)**
   $\forall x$ Likes (x,sweets)
9. **Define resolution.**
   - Resolution is a procedure which gains efficiency from the fact that it operates on statements that have been converted to a very convenient standard form.
   - It is a simple iterative process in which two parent clauses are compared, yielding a new clause that has been inferred from them.
10. **State Herbrand's theorem.**
    - To show that a set of clauses S is unsatisfiable, it is necessary to consider only interpretations over a particular set, called the Herbrand universe of S.

- A set of clauses S is unsatisfiable if and only if a finite subset of ground instances (in which all bound variables have had a value substituted for them) of S is unsatisfiable.

11. State the use of unification. (MAY/JUNE 2012)

What is the significance in using the unification algorithm?(NOV/DEC 2012)

In order to apply the rules of inference, an inference system must be able to determine when two expressions match. Unification used for determining the substitutions needed to make two predicate calculus expressions match.

- All variables must be universally quantified.
- Existentially quantified variables may be eliminated by replacing them with constants that make the sentence true.

For example, in ∃X mother(X,bill), we can replace X with a constant designating bill's mother, ann, to get: mother(ann, bill).

The UNIFY algorithm takes two sentences and returns a unifier for them if one exists:

UNIFY(p,q)=θ where SUBST(θ,p)=SUBST(θ,q)

The problem arises only because two sentences happen to use the same variable name x. the problem can be avoided by standardizing apart one of the two sentences being unified, which means renaming its variables to avoid name clashes.

12. Is games appeared to be a good domain to explore machine intelligence? Justify.

Yes, Games appeared to be a good domain to explore machine intelligence for the following reasons:

- They provide the structured task I which it is very easy to measure success of failure.
- They did not obviously require large amounts of knowledge for simpler games.

13. What is MINIMAX Search Procedure?

The MINIMAX 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.

14. What is Alpha-Beta pruning?

The problem with minimax search is that the number of game states it has to examine is exponential in the number of moves. Unfortunately, exponent cannot be eliminated but can be effectively cut it in half. By performing pruning, large part of the tree can be eliminated from consideration.

- **α** : the value of the best(i.e.,highest-value) choice we have found so far at any
- **β:** the value of best (i.e., lowest-value) choice we have found so far at any choice point along the path of MIN.

15. Write Depth-First Iterative Deepening.

a. Set SEARCH-DEPTH=1
b. Conduct a depth-first search to a depth of SEARCH-DEPTH. If the solution path is found, then return it.
c. Otherwise, increment SEARCH-DEPTH by 1 and go to step b.

16. Write Iterative-Deepening –A* Algorithm.

a. Set THRESHOLD=the heuristic evaluation of the start state.
b. Conduct a depth-first search, pruning any branch when its total cost function (g+h') exceeds THRESHOLD. If the solution path is found during the search, return it.
c. 'Otherwise, increment THRESHOLD by the minimum amount it was exceeded during the previous step, and then go to step b.

17. Give some examples of structured representation of knowledge.

- Semantic Nets
- Frames
- Scripts
- Conceptual dependency
- CYC

s

18. **Define Semantic Nets.**
    In Semantic Net, information is represented as a set of nodes connected to each other by a set of labeled arcs, which represent relationships among the modes.
19. **What are frames?**
    A frame is a collection of attributes and associated values that describe some entity in the world.
20. **What is frame language?**
    The idea of a frame system as a way to represent declarative knowledge has been encapsulated in a series of frame-oriented knowledge representation languages, whose features have evolved and been driven by an increased understanding of the sort of representation issues.
    Eg. KRL,FRL,KRYPTON,CYCL etc.
21. **Define Conceptual Dependency.**
    Conceptual Dependency is a theory of how to represent the kind of knowledge about events that is usually contained in natural language sentences.
22. **State the goal of Conceptual dependency.**
    The goal is to represent the knowledge in a way that,
    - Facilitates drawing inferences from the sentences.
    - Is independent of the language in which the sentences were originally stated.
23. **What are scripts?**
    A script is a structure that describes the stereotyped sequence of events in a particular context. A script consists of a set of slots and associated with each slot may be some information about what kind of values it may contain as well as a default value to be used if no other information is available.
24. **What is CYC?**
    CYC is a very large knowledge base project aimed at capturing human commonsense knowledge. The goal of CYC is to encode the large body of knowledge that is so obvious that it is easy to forget to state it explicitly.
25. **What are the fundamental problems of Structured knowledge representation?**
    - Knowledge acquisition: Learning a schema system is long, tedious, and ad hoc process.
    - Context Recognition (The Frame problem): Many problems are easily solved when the context is known. Recognizing the correct context can be very difficult.

PART-B

1. Explain in detail the approaches to Knowledge Representation.

Approaches To Knowledge Representation

A good system for the representation of structured knowledge in a particular domain should possess the following four properties:

(i) **Representational Adequacy**:- The ability to represent all kinds of knowledge that are needed in that domain.

(ii) **Inferential Adequacy** :- The ability to manipulate the represented structure to derive new structures corresponding to the new knowledge inferred from old..

(iii) **Inferential Efficiency**:- The ability to incorporate additional information into the knowledge structure that can be used to focus the attention of the inference mechanisms in the most promising directions.

(iv) **Acquisitional Efficiency** :- The ability to acquire new knowledge using automatic methods wherever possible rather than reliance on human intervention.

There are four types of knowledge representation

- Relational Knowledge
- Inheritable Knowledge
- Inferential Knowledge

- Procedural Knowledge

(i) Relational Knowledge

Relational knowledge provides a frame work to compare two objects based on equivalent attributes. This knowledge associates element of one domain with another domain. Relational knowledge is made up of objects consisting of attributes as their corresponding associated values. The results of this knowledge type is mapping of elements among different domains.

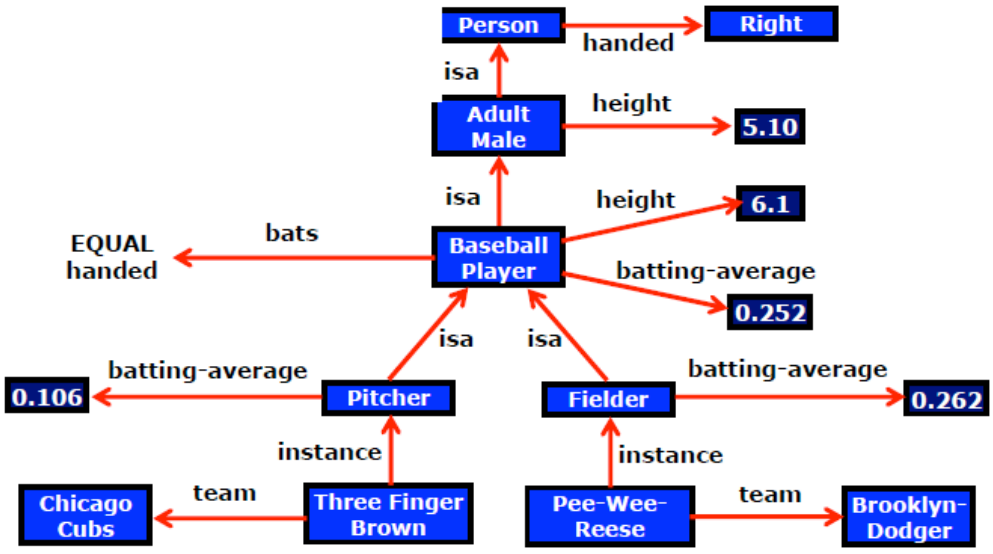
The table below shows a simple way to store facts. The facts about the set of objects are put systematically in columns.

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 fact that it is not [possible to answer a simple question such as "Who is the heaviest player?". But, if the procedure for finding the heaviest player is provided, then these facts will be enable that procedure to compute an answer. This representation provides only little opportunity for inference.

(ii) Inheritable Knowledge

Inheritable knowledge is obtained from associated objects. It describes a structure in which new objects are created which may inherit all or subset of attributes from existing objects.



Inheritable knowledge representation

The directed arrow represent attributes originates at object being described and terminates at object or its value. The box nodes represent objects and values of the attributes.

Viewing node as a frame: Baseball Player

isa :Adult-Male
bats : Equal to handed
height : 6.1

bating average:0.252

Algorithm: Property of Inheritance

The steps for retrieving a value v for an attribute A of an instance object O

i) Find object O in the knowledge base.

ii) If there is a value for the attribute A, then report that value.

iii) Otherwise, see if there is a value for the attribute instance. If not then fail.

iv) Otherwise, move to the node corresponding to that value and look for a value for the attribute A. If one is found, report it.

v) Otherwise, do until there is no value for the isa attribute or until an answer is found.

(a) Get the value of "isa" attribute and move to that node.

(b) See if there is a value for the attribute A; If yes, report it.

Team(Pee-Wee-Reese)=Brooklyn Dodger

Batting average(Three finger Brown)=0.106

Height(Pee-Wee-Reese)=6.1

Bats(Three finger Brown)=right

iii) Inferential Knowledge

Inferential knowledge is inferred from objects through relations among objects. 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 with in linguistic is called semantics.

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

Example

Given a set of values and relations, one may infer other values or relations. Predicate logic is used to infer from a set of attributes. Inference from a predicate logic uses a set of logical operations to relate individual data.

i) Wonder is a name of a dog.

Dog(Wonder)

ii) All dogs belong to the class of animals.

$\forall x: dog(x) \rightarrow animal(x)$

iii) All animals either live on land or in water.

$\forall x: animal(x) \rightarrow live(x, land) \vee live(x, water)$

From these three statements we infer that

Wonder lives either on land or on water

iv) Procedural Knowledge

Procedural Knowledge specifies what to do when. The procedural knowledge is represented in program in many ways. The machine uses the knowledge when it executes the code to perform a task.

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

2. Explain the various issues in knowledge Representation in detail.

The fundamental goal of knowledge representation is to facilitate inferencing from knowledge. The issues that arise while using knowledge representation techniques are many. Some of these are explained below.

- Any attributes of objects so basic that they appear in almost every problem domain?. If there are we need to make sure that they are handled appropriately in each of the mechanisms we propose.
- Any important relationship that exists among object attributes?
- At what level of detail should the knowledge be represented?
- How sets of objects be represented?
- Given a large amount of knowledge stored in a database, how can relevant parts be accessed?

i) Important Attributes

There are two attributes that are of very general significance

- Isa
- Instance

These attributes are important because they support property inheritance. They represent class membership and class inclusion and the class inclusion is transitive

ii) Relationships among Attributes

The attributes that are used to describe objects are themselves entities. The relationship between the Each entity have four properties independent of the specific knowledge they encode. They are:

(a) Inverses

(b) Existence in an isa hierarchy

(c) Techniques for reasoning about values

(d) Single-valued attributes

(a) Inverses:

Entities in the world are related to each other in many different ways. Relationships are attributes such as **isa, instance and team.** There are two good ways to represent the relationship.

The first is to represent both relationships in a single representation

team(Pee-Wee-Reese, Brooklyn-Dodgers) can equally easily be represent as a statement about Pee-Wee-Reese or about the Brooklyn-Dodgers.

The second approach is to use attributes that focus on a single entity but to use them in pairs, one the inverse of the other. In this approach, represent the team information with two attributes:

team = Brooklyn-Dodgers

team-members = Pee-Wee-Reese;…

This is the approach that is taken in semantic net and frame-based systems. Each time a value is added to one attribute then the corresponding value is added to the inverse.

(b) Existence in an isa hierarchy

The attribute height is the specialization of general attribute physical size which is in turn a specialization of physical attribute. The generalization specialization attributes are important because they support inheritance.

(c) Techniques for reasoning about values

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

- Information about the type of value. Eg height must be a unit of length.
- Constraints on the value. Eg age of a person cannot be greater than the age of persons parents.
- Rules for computing the value when it is needed. Eg bats attribute. These rules are called backward rules. Such rules are also called if-needed rules.

- Rules that describe actions that should be taken if a value ever becomes known. These rules are called forward rules or sometimes if-added rules.

(d) Single-valued attributes

This is about specific attribute that is guaranteed to take unique value. Eg. A baseball player can at a time have only a single height and be a member of only one team.

(iii) Choosing the granularity of representations

It deals with what level should the knowledge be represented. The primitives are

- 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, Consider the following fact

John spotted Smith

This could be represented as

Spotted(John, Smith)

Or

Spotted(agent(John), object(smith))

Such representation would make it easy to answer question such as

Who spotted Smith?

Suppose we want to know

Did John see smith?

Given only one fact, but we cannot discover the answer. So we can add another fact

Spotted(x, y)→ saw(x, y)

We can now infer the answer to the question

(iv)Representing set of objects

Certain properties of objects that is true as member of a set but not as individual.

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 to attach assertion to the sets representing people, sheep and English. The reason to represent sets of object 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 explicitly with every elements of the set.This is done in different ways:

- In logical representation through the use of universal quantifier,
- In hierarchical structure where node represents 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.
- Asserting some property that holds for individual elements of the set, means, anything that is an elephant is large.

There are three ways in which sets may be represented

- Names
- Extensional definition
- Intentional definition

(v) Finding the right structures as needed

To access the right structure for describing a particular situation, it is necessary to solve all the following problems,

- How to perform an initial selection of the most appropriate structure
- How to fill in appropriate details from the current situation

- 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 structure is appropriate
- When to create and remember a new structure
    i. Selecting an Initial structure
        - Index the structure
        - Pointer to all the structures
        - Locate one major due in the problem description
    ii. Revising the choice when necessary

3. Differentiate predicate and propositional logic. Explain predicate logic with suitable illustrations.

Sl.No	Predicate logic	Propositional logic
1.	Predicate logic is a generalization of propositional logic that allows us to express and infer arguments in infinite models.	A proposition is a declarative statement that's either TRUE or FALSE (but not both).
2.	Predicate logic (also called predicate calculus and first-order logic) is an extension of propositional logic to formulas involving terms and predicates. The full predicate logic is undecidable	Propositional logic is an axiomatization of Boolean logic. Propositional logic is decidable, for example by the method of truth table
3.	Predicate logic have variables	Propositional logic has variables. Parameters are all constant
4.	A predicate is a logical statement that depends on one or more variables (not necessarily Boolean variables)	Propositional logic deals solely with propositions and logical connectives
5.	Predicate logic there are objects, properties, functions (relations) are involved	Proposition logic is represented in terms of Boolean variables and logical connectives
6.	In predicate logic, we symbolize subject and predicate separately. Logicians often use lowercase letters to symbolize subjects (or objects) and uppercase letter to symbolize predicates.	In propositional logic, we use letters to symbolize entire propositions. Propositions are statements of the form "x is y" where x is a subject and y is a predicate.
7.	Predicate logic uses quantifiers such as universal quantifier ("∀"), the existential quantifier ("∃")	Prepositional logic has no quantifiers.
8.	**Example** Everything is green" as "∀x Green(x)"	**Example** Everything is green" as "G(x)"

	or "Something is blue" as "∃x Blue(x)".	or "Something is blue" as "B(x)".

Predicate logic

FOL or predicate logic is a generalization of propositional logic that allows us to express and infer arguments in infinite models like

iii. All men are mortal
iv. Some birds cannot fly

Predicates are like functions except that their return type is true or false.

A predicate with no variable is a proposition.

Syntax of First-Order Logic

- Constants KingJohn, 2, …
- Predicates/Relation Brother, >, …
- Functions Sqrt, LeftArmOf, …
- Variables x, y, a, b, …
- Connectives ∧ ∨ ¬ ⇒ ⇔
- Equality =
- Quantifiers $ "

Components of First-Order Logic

- **Term**
  - Constant, e.g. Red
  - Function of constant, e.g. Color(Block1)
- **Atomic Sentence**
  - Predicate relating objects (no variable)
    - Brother (John, Richard)
    - Married (Mother(John), Father(John))
- **Complex Sentences**
  - Atomic sentences + logical connectives
    - Brother (John, Richard) ∧¬Brother (John, Father(John))
- **Quantifiers**
  - Each quantifier defines a variable for the duration of the following expression, and indicates the truth of the expression…
- **Universal quantifier "for all" ∀**
  - The expression is true for every possible value of the variable
- **Existential quantifier "there exists" ∃**
  - The expression is true for at least one value of the variable

Examples

1. Some dogs bark.

∃x dog(x) ∧ bark(x)

2. All dogs have four legs.

∀x(dog(x)→have_four_legs(x))

(or)

∀x(dog(x)→legs(x,4))

3. All barking dogs are irritating

$\forall x(dog(x) \wedge barking(x) \rightarrow irritating(x))$

4. No dogs purr.

$\neg \exists x\ (dog(x) \wedge purr(x))$

Problem on Resolution using predicate logic

1. All people who are graduating are happy.

2. All happy people smile.

3. Someone is graduating.

4. Conclusion: Is someone smiling?

Solution:

Convert in to predicate Logic

1. $\forall x[graduating(x) \rightarrow happy(x)$
2. $\forall x(happy(x) \rightarrow smile(x)$
3. $\exists x$ graduating(x)
4. $\exists x$ smile(x)

Convert to clausal form

(i) Eliminate the → sign

1. $\forall x \neg$ graduating(x) $\vee$happy(x)
2. $\forall x \neg$happy(x) $\vee$smile(x)
3. $\exists x$ graduating(x)
4. $\neg \exists x$ smile(x)

(ii) Reduce the scope of negation

1. $\forall x \neg$ graduating(x) $\vee$happy(x)
2. $\forall x \neg$happy(x) $\vee$smile(x)
3. $\exists x$ graduating(x)
4. $\forall x \neg$ smile(x)

(iii) Standardize variables apart

1. $\forall x \neg$ graduating(x) $\vee$happy(x)
2. $\forall y \neg$happy(y) $\vee$smile(y)
3. $\exists x$ graduating(z)
4. $\forall w \neg$ smile(w)

(iv) Move all quantifiers to the left

1. $\forall x \neg$ graduating(x) $\vee$happy(x)
2. $\forall y \neg$happy(y) $\vee$smile(y)
3. $\exists x$ graduating(z)
4. $\forall w \neg$ smile(w)

(v) Eliminate ∃

1. $\forall x \neg$ graduating(x) $\vee$happy(x)
2. $\forall x \neg$happy(y) $\vee$smile(y)
3. graduating(name1)
4. $\forall w \neg$ smile(w)

(vi) Eliminate ∀

1. $\neg$ graduating(x) $\vee$happy(x)
2. $\neg$happy(y) $\vee$smile(y)

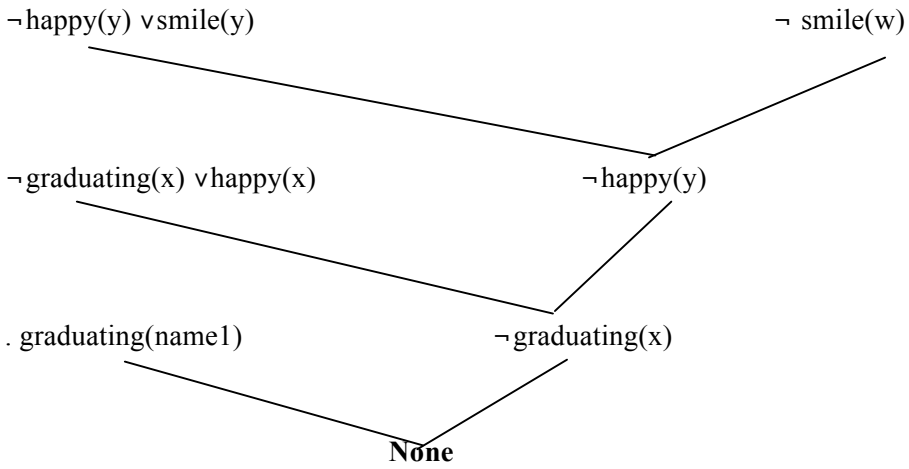
3. graduating(name1)

4. ¬ smile(w)

(vii) Convert to conjunct of disjuncts form.

(viii) Make each conjunct a separate clause.

(ix) Standardize variables apart again.



Thus, we proved someone is smiling.

4. Explain the unification algorithm used for reasoning under predicate logic with an example. (APRIL/MAY 2011)

When attempting to match 2 literals, all substitutions must be made to the entire literal. There may be many substitutions that unify 2 literals, the most general unifier is always desired

1. Initial predicate symbols must match.
2. For each pair of predicate arguments:

– different constants cannot match.

– a variable may be replaced by a constant.

– a variable may be replaced by another variable.

– a variable may be replaced by a function as long as the function does not contain an instance of the variable

Algorithm :Unify(L1,L2)

1. If L1 or L2 are both variables or constants, then:
    (a) If L1 and L2 are identical, then return NIL.
    (b) Else if L1 is a variable, then if L1 occurs in L2 the return {FAIL}, else return (L2/L1).
    (c) Else if L2 is a variable, then if L2 occurs in L1 the return {FAIL}, else return (L1/L2).
    (d) Else return {FAIL}
2. If the initial predicate symbols in L1 and L2 are not identical, the return {FAIL}.
3. If L1 and L2 have a different number of arguments, then return {FAIL}.
4. Set SUBST to NIL.
5. For i←1 to number of arguments in L1:
    (a) Call Unify with the i^{th} argument of L1 and the i^{th} argument of L2, putting result in S.
    (b)If s contains FAIL then return {FAIL}
    (c) If S is not equal to NIL then:
        (i) Apply S to the remainder of both L1 and L2
        (ii) SUBST=APPEND(S,SUBST).
6. Return SUBST.

Example

P(x) and P(y) :substitution = (x/y)

• P(x,x) and P(y,z) : (z/y)(y/x)

• P(x,f(y)) and P(Joe,z) : (Joe/x, f(y)/z)

• P(f(x)) and P(x) : can't do it!

• P(x) Ú Q(Jane) and P(Bill) ∨ Q(y): (Bill/x, Jane/y)

5. Consider the following facts

b. Team India

c. Team Australia

d. Final match between India and Australia

e. India scored 350 runs, Australia scored 350 runs, India lost 5 wickets, Australia lost 7 wickets.

f. The team which scored the maximum runs wins.

g. If the scores are same the team which lost minimum wickets wins the match.

Represent the facts in predicate, convert to clause form and prove by resolution "India wins the match". (NOV/DEC 2011)

Solution:

Convert in to predicate Logic

(a) team(India)

(b) team(Australia)

(c) team(India) ∧ team(Australia)→final_match(India, Australia)

(d) score(India,350) ∧ score(Australia,350) ∧wicket(India,5) ∧ wicket(Australia,7)

(e) ∃x team(x) ∧ wins(x)→score(x, max_runs))

(f) ∃xy score(x,equal(y)) ∧ wicket(x, min) ∧final_match(x,y)→win(x)

Convert to clausal form

(i) Eliminate the → sign

(a) team(India)

(b) team(Australia)

(c) ¬ (team(India) ∧ team(Australia)) ∨ final_match(India, Australia)

(d) score(India,350) ∧ score(Australia,350) ∧wicket(India,5) ∧ wicket(Australia,7)

(e) ∃x¬ (team(x) ∧ wins(x)) ∨score(x, max_runs))

(f) ∃xy¬ (score(x,equal(y)) ∧ wicket(x, min) ∧final_match(x,y)) ∨win(x)

(ii) Reduce the scope of negation

(a) team(India)

(b) team(Australia)

(c) ¬ team(India) ∨ ¬team(Australia) ∨ final_match(India, Australia)

(d) score(India,350) ∧ score(Australia,350) ∧wicket(India,5) ∧ wicket(Australia,7)

(e) ∃x ¬ team(x) ∨ ¬wins(x) ∨score(x, max_runs))

(f) ∃xy¬ score(x,equal(y)) ∨¬wicket(x, min_wicket) ∨¬final_match(x,y)) ∨win(x)

(iii) Standardize variables apart

(iv) Move all quantifiers to the left

(v) Eliminate ∃

(a) team(India)

(b) team(Australia)

(c) ¬ team(India) ∨ ¬team(Australia) ∨ final_match(India, Australia)

(d) score(India,350) ∧ score(Australia,350) ∧wicket(India,5) ∧ wicket(Australia,7)

(e) ¬ team(x) ∨ ¬wins(x) ∨score(x, max_runs))

(f) ¬ score(x,equal(y)) ∨¬wicket(x, min_wicket) ∨¬final_match(x,y)) ∨win(x)

(vi) Eliminate ∀

(vii) Convert to conjunct of disjuncts form.

(viii) Make each conjunct a separate clause.

(a) team(India)

(b) team(Australia)

(c) ¬ team(India) ∨ ¬team(Australia) ∨ final_match(India, Australia)

(d) score(India,350)

score(Australia,350)

wicket(India,5)

wicket(Australia,7)

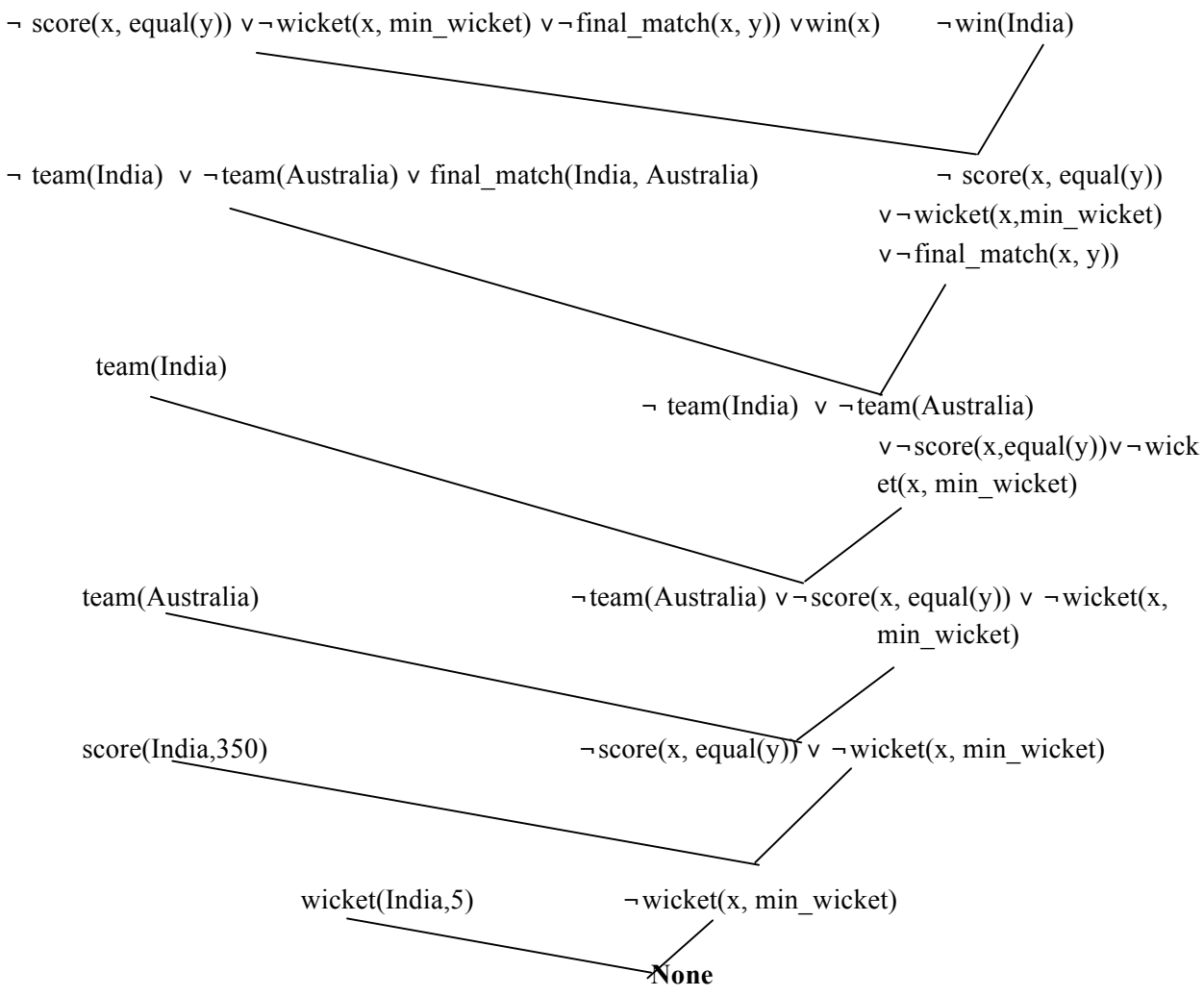
(e) ¬ team(x) ∨ ¬wins(x) ∨score(x, max_runs))

(f) ¬ score(x, equal(y)) ∨¬wicket(x, min_wicket) ∨¬final_match(x, y)) ∨win(x)

(ix) Standardize variables apart again.

Prove: win(India)

Disprove: ¬win(India)



Thus, proved India wins match.

6. Consider the following facts and represent them in predicate form:

F1. There are 500 employees in ABC company.

F2. Employees earning more than Rs. 5000 pay tax.

F3. John is a manager in ABC company.

F4. Manager earns Rs. 10,000.

Convert the facts in predicate form to clauses and then prove by resolution: "John pays tax". (NOV/DEC 2012)

Solution:

Convert in to predicate Logic

1. company(ABC) ∧employee(500,ABC)
2. ∃x company(ABC) ∧employee(x, ABC) ∧earns(x,5000)→pays(x, tax)
3. manager(John, ABC)
4. ∃x manager(x, ABC)→earns(x,10000)

Convert to clausal form

(i) Eliminate the → sign

1. company(ABC) ∧employee(500,ABC)
2. ∃x¬ (company(ABC) ∧employee(x, ABC) ∧earns(x,5000)) ∨ pays(x, tax)
3. manager(John, ABC)
4. ∃x ¬manager(x, ABC) ∨ earns(x, 10000)

(ii) Reduce the scope of negation

1. company(ABC) ∧employee(500,ABC)
2. ∃x¬ company(ABC) ∨¬employee(x, ABC) ∨¬earns(x,5000) ∨ pays(x, tax)
3. manager(John, ABC)
4. ∃x ¬manager(x, ABC) ∨ earns(x, 10000)

(iii) Standardize variables apart

1. company(ABC) ∧employee(500,ABC)
2. ∃x¬ company(ABC) ∨¬employee(x, ABC) ∨¬earns(x,5000) ∨ pays(x, tax)
3. manager(John, ABC)
4. ∃y ¬manager(x, ABC) ∨ earns(y, 10000)

(iv) Move all quantifiers to the left

(v) Eliminate ∃

1. company(ABC) ∧employee(500,ABC)
2. ¬ company(ABC) ∨¬employee(x, ABC) ∨¬earns(x,5000) ∨ pays(x, tax)
3. manager(John, ABC)
4. ¬manager(x, ABC) ∨ earns(y, 10000)

(vi) Eliminate ∀

(vii) Convert to conjunct of disjuncts form.

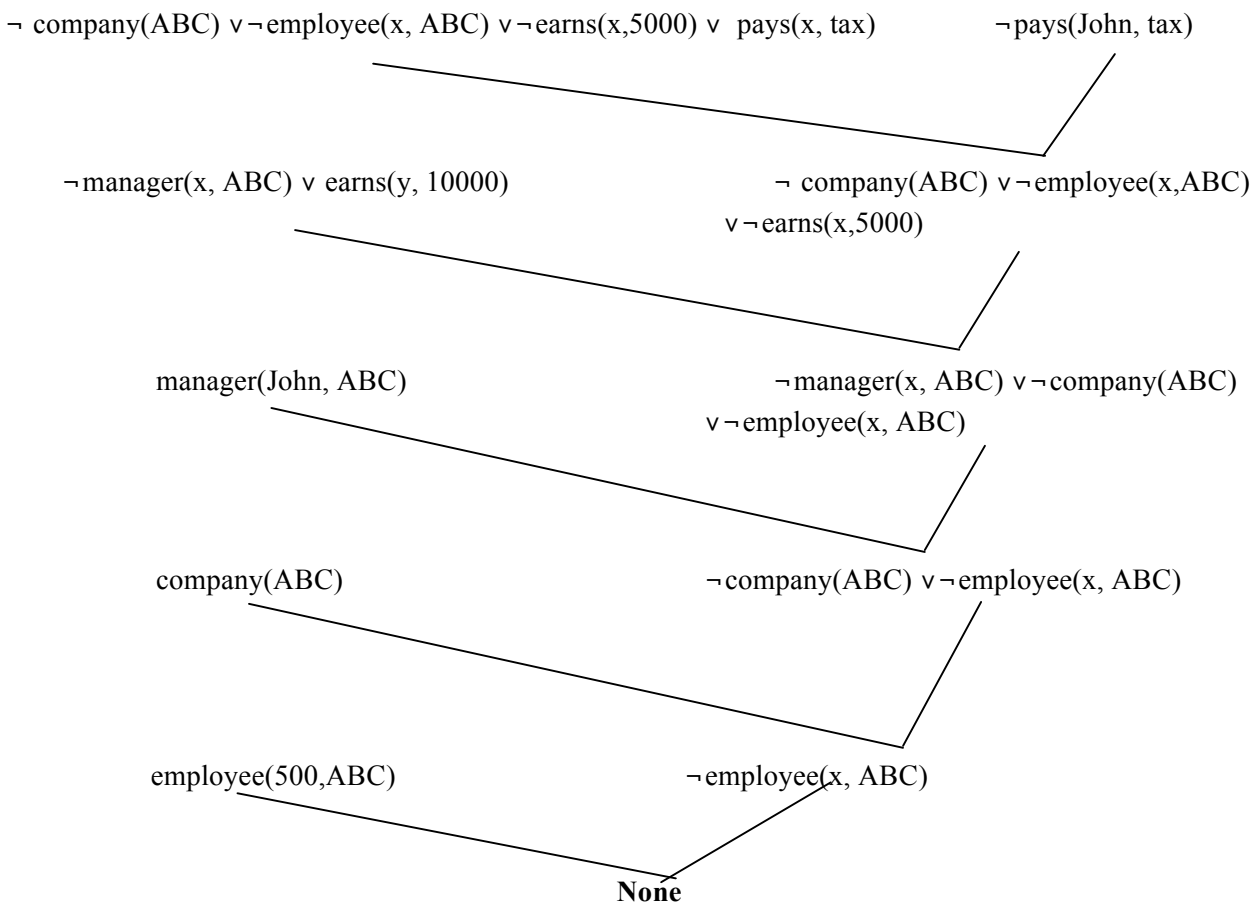
(viii) Make each conjunct a separate clause.

1. (a)company(ABC)
   (b)employee(500,ABC)
2. ¬ company(ABC) ∨¬employee(x, ABC) ∨¬earns(x,5000) ∨ pays(x, tax)
3. manager(John, ABC)
4. ¬manager(x, ABC) ∨ earns(y, 10000)

(ix) Standardize variables apart again.

Prove :pays(John, tax)

Disprove: ¬pays(John, tax)



Thus, proved john pays tax.

7. Explain with an example concept of resolution. (NOV/DEC 2012)

Resolution is a procedure used in proving that arguments which are expressible in predicate logic are correct. Resolution is a procedure that produces proofs by refutation or contradiction. Resolution leads to refute a theorem-proving technique for sentences in propositional logic and first order logic.

i. Resolution is a rule of inference
ii. Resolution is a computerized theorem prover

Algorithm: Propositional Resolution

Let F be a set of axioms and P the theorem to prove.

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

(a) Select two clauses. Call these the parent clauses.

(b) 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 pair 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.

(c) 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.

Example

1. If a triangle is equilateral then it is isosceles.
2. If a triangle is isosceles the two sides AB and AC are equal.
3. If AB and AC are equal then angle B and angle C are equal.
4. ABC is an Equilateral triangle.

Prove "Angle B is equal to angle C"

Propositional Logic

4. Equilateral(ABC)
1. Equilateral(ABC)→Isosceles(ABC)
2. Isosceles(ABC)→Equal(AB,AC)
3. Equal(AB,AC)→ Equal(B,C)

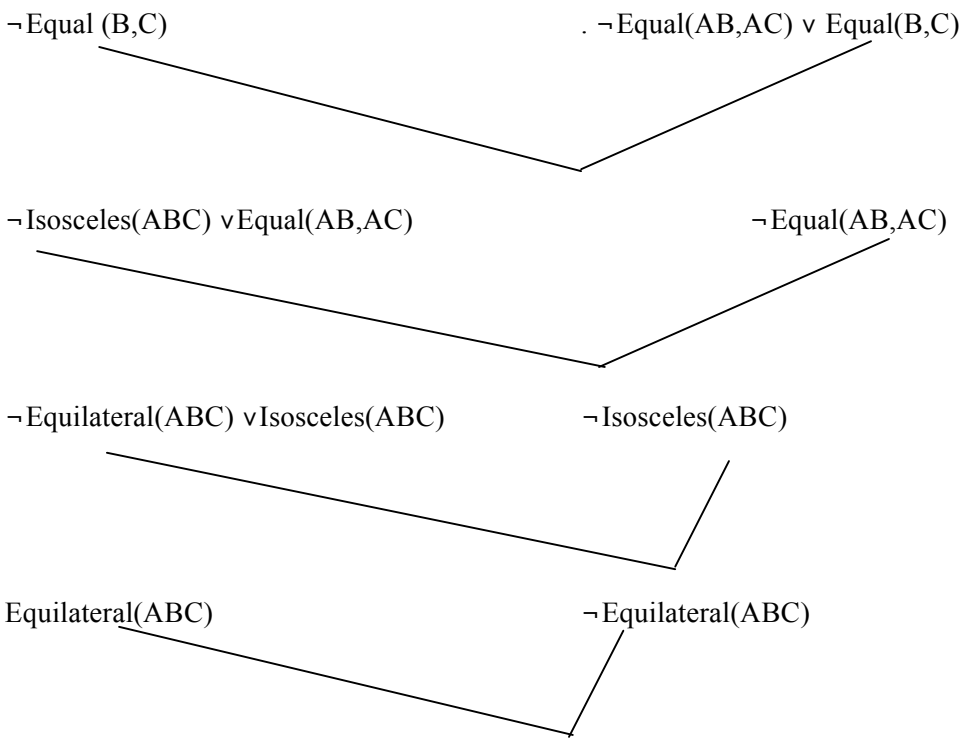
Convert to clausal form

1. ¬Equilateral(ABC) ∨Isosceles(ABC)
2. ¬Isosceles(ABC) ∨Equal(AB,AC)
3. ¬Equal(AB,AC) ∨ Equal(B,C)
4. Equilateral(ABC)

To prove " Equal (B,C)"

Let us disprove "Not equal B and C

¬Equal (B,C)"



None

Thus we proved that angle B is equal to angle C

8. Explain MINIMAX Search Procedure algorithm with suitable illustration.

MINIMAX Search Algorithm

It is a depth first, depth limited search procedure. The idea is to start at the current position and uses a plausible move generator to generate the set of possible successor positions. Apply static evaluation function to those positions and simply choose the best one. After doing so, we can back that value up to the starting position to represent the evaluation of it. We assume that the static evaluation function returns large values to indicate good situation so our goal is to maximize the value of the static function for the next board position

Principle of Minimax

Given two players(x,y)- the chance of one person winning the game is possible at the expense of other person losing the game.

One ply search

- Maximize the value
- Selecting B's value to A, conclude that A's value is 8.

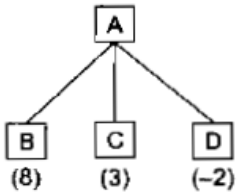
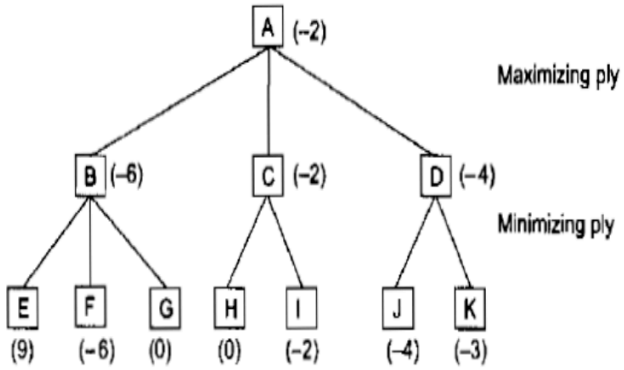


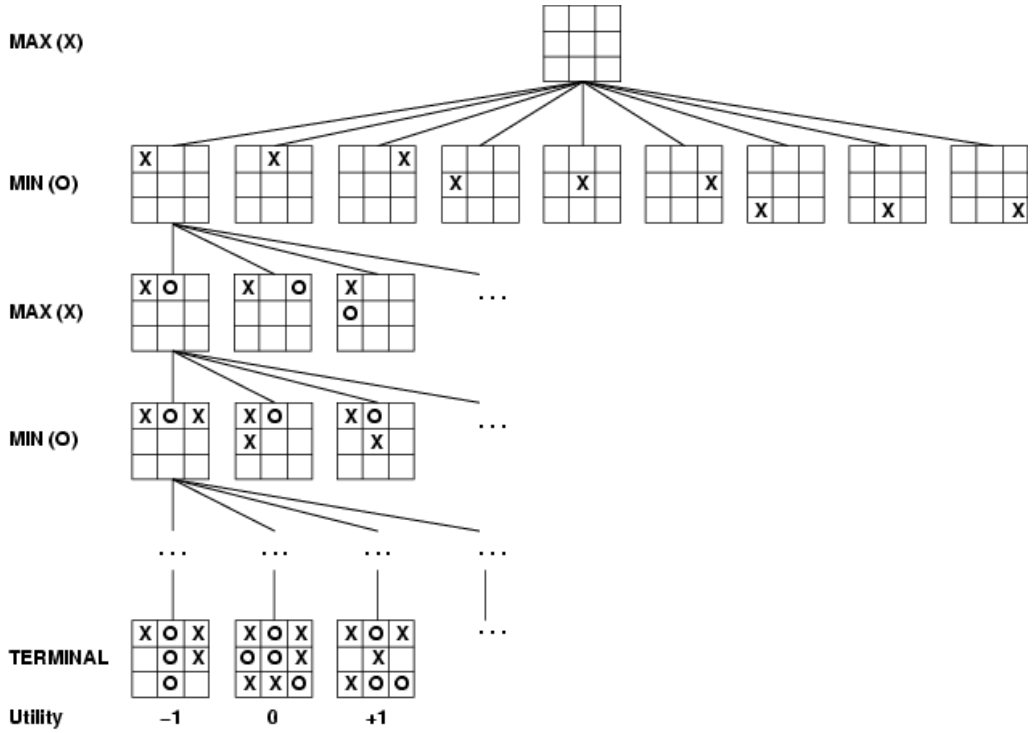
Fig. 12.1 *One-Ply Search*

Two ply search



Backing Up the Values of a Two-Ply Search

The alternation of maximizing and minimizing at alternate ply when evaluations being pushed back up correspond to the opposing strategies of two plays and gives this method a name minimax procedure.



The computer is **Max**.

The opponent is **Min**.**Minimax procedure**

- Create start node as a MAX node with current board configuration
- Expand nodes down to some **depth** (a.k.a. **ply**) of lookahead in the game
- Apply the evaluation function at each of the leaf nodes
- "Back up" values for each of the non-leaf nodes until a value is computed for the root node
    - At MIN nodes, the backed-up value is the **minimum** of the values associated with its children.
    - At MAX nodes, the backed up value is the **maximum** of the values associated with its children.
- Pick the operator associated with the child node whose backed-up value determined the value at the root

The important functions in the minimax algorithm are

- MOVEGEN (position, player) – The plausible move generator which return a list of nodes representing the moves that can be made by player in position.
- STATIC(position, player) – Static evaluation function
- DEEP-ENOUGH (position, depth) – It evaluates and return TRUE if the search should be stopped at the current level and FALSE otherwise.

The MNIMAX procedure has to return two results

- The back up value of the path it chooses (VALUE)
- The path itself (PATH)

Algorithm

1. If DEEP-ENOUGH (position, depth), then return structure.

    VALUE=STATIC (position, player)

    PATH=Nil

This indicates that there is no path from this node.

2. Otherwise generate one more ply of the tree by calling the function MOVE-GEN (position, player) and setting SUCCESSORS to the list it returns.

3. If SUCCESSORS is empty, then there are no moves to be done, so return the same structure that would have been returned by DEEP_ENOUGH.

4. If SUCCESSORS is not empty, then examine each element in turn and keep track of the best one. This is done as follows.

Initialize BEST-SCORE to the minimum value.

For each element SUCC of SUCCESSORS do the following

(a) Set RESULT-SUCC to MINIMAX (SUCC, Depth+1, OPPOSITE (player)

(b) Set NEW-VALUE to VALUE (RSULT-SUCC)

(c) If NEW-VALUE > BEST-SCORE, then we have found the successor that is better than any that have been examined so far. Record this by doing the following.

(i) Set BEST-SCORE to NEW VALUE.

(ii) The best known path is now from CURRENT to SUCC and then on the appropriate path down from SUCC as determined by the recursive call to MINIMAX.

5. Now that all successors have been examined, we know value of position as well as which path to take from it. So return the structure.

VALUE = BEST-SCORE

PATH = BEST-PATH

Performance

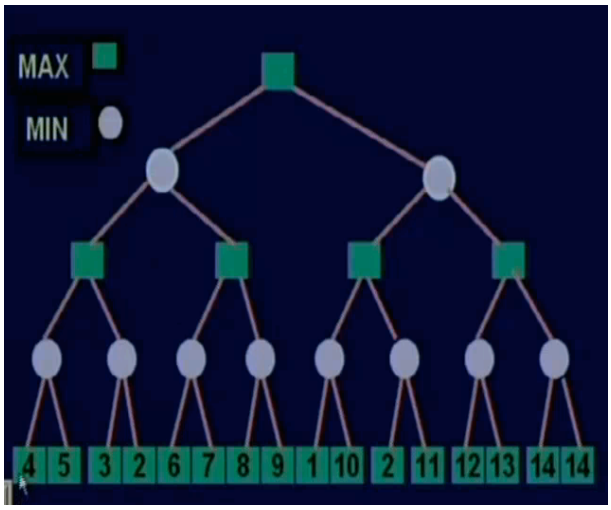
If the maximum depth of the tree is m, and there are b legal moves at each point, then the time complexity of the minimax algorithm is O (b^m).

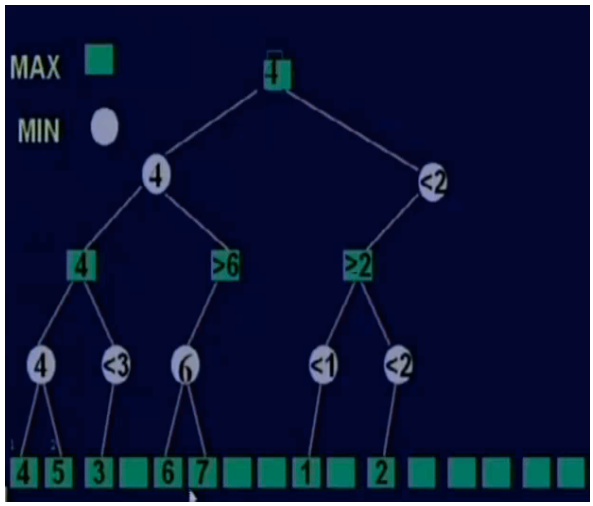
9. Explain alpha-beta pruning in detail along with example.

Alpha Beta Cut-off

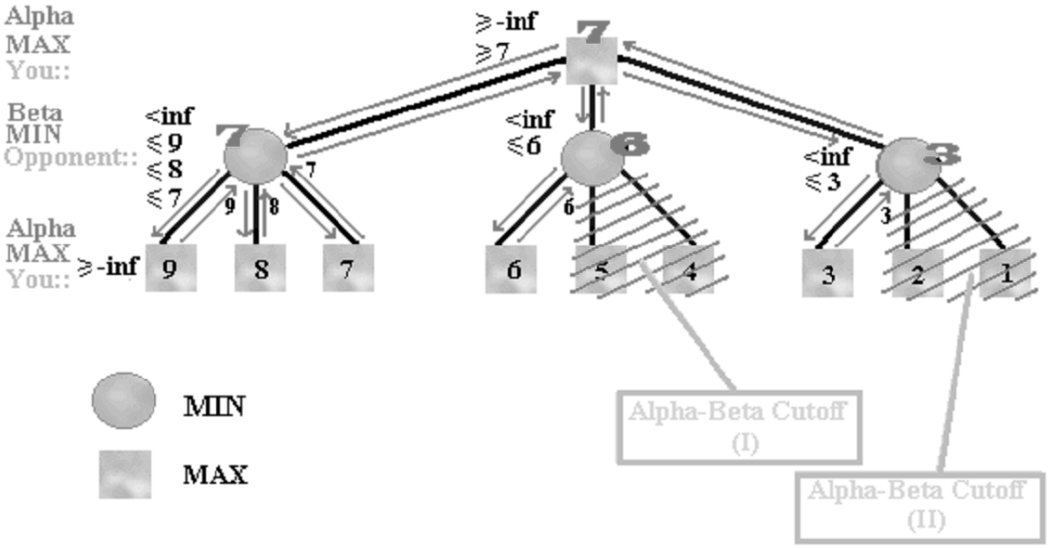
MINIMAX searches the entire tree, even if in some cases the rest can be ignored. But, this alpha beta cutoff returns appropriate minimax decision without exploring entire tree. The minimax search procedure is slightly modified by including branch and bound strategy one for each players. This modified strategy is known as alpha beta pruning. It requires maintenance of two threshold values, one representing a lower bound on the value that a maximizing node may ultimately be assigned (alpha) and another representing upper bound on the value that a minimizing node may be assigned (beta).

Here is an example of Alpha-Beta search:





Another example



Algorithm: MINIMAX-A-B(position, depth, player, Use-Thresh, Pass-Thresh)

1. If DEEP-ENOUGH (position, depth), then return structure.

VALUE=STATIC (position, player)

PATH=Nil

This indicates that there is no path from this node.

2. Otherwise generate one more ply of the tree by calling the function MOVE-GEN (position, player) and setting SUCCESSORS to the list it returns.

3. If SUCCESSORS is empty, then there are no moves to be done, so return the same structure that would have been returned by DEEP_ENOUGH.

4. If SUCCESSORS is not empty, then examine each element in turn and keep track of the best one. This is done as follows.

For each element SUCC of SUCCESSORS do the following

(a) Set RESULT-SUCC to MINIMAX-A-B (SUCC, Depth+1, OPPOSITE (player), Pass-Thresh, Use-Thresh)

(b) Set NEW-VALUE to VALUE (RSULT-SUCC)

(c) If NEW-VALUE > Pass-Thresh, then we have found a successor that is better than any that have been examined so far. Record this by doing the following.

(i) Set Pass-Thresh to NEW VALUE.

(ii) The best known path is now from CURRENT to SUCC and then on the appropriate path down from SUCC as determined by the recursive call to MINIMAX-A-B. So set BEST-PATH to the result of attaching SUCC to the front of PATH (RESULT-SUCC).

5. If Pass-Thresh is not better than Use-Thresh, then we should stop examining this branch. But, both thresholds and values have been inverted. So, if Pass-Thresh>=Use-Thresh, then return immediately with the value.

VALUE = Path-Thresh

PATH = BEST-PATH

5. So return the structure.

VALUE = Path-Thresh

PATH = BEST-PATH

Performance analysis

- Guaranteed to compute the same root value as MINIMAX.
- If we choose a best successor first, the need to examine $O(b^{m/2})$ nodes.
- If we choose a random successor, the need to examine $O(b^{3m/4})$ nodes.

10. Explain various structured knowledge representations in detail.

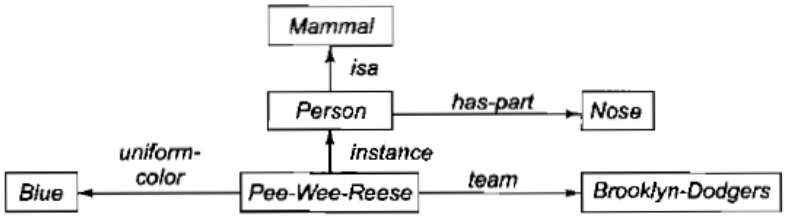
Modeling-based representations reflect the structure of the domain, and then reason based on the model.

- **Semantic Nets**
- **Frames**
- **Conceptual Dependency**
- **Scripts**
- **CYC**

(i) Semantic Networks

In semantic net, information is represented as a set of nodes connected to each other by a set of labelled arcs, which represent relationship among the nodes. In this network, inheritance is used to derive the additional relation.

Eg has_part(Pee-Wee-Reesse, nose)



Intersection search

To find the relationship among objects by spreading activation out from each of two nodes and seeing where the activation function met. This process is called intersection search.

Example:

What is the connection between the Brooklyn dodger and blue?

The answer is Pee-Wee-Reese

Representing non binary predicates

Semantic nets are a natural way to represent relationships that would appear as binary predicates.

instance(pee-Wee-Reese, Person)

team(Pee-Wee-Reese, Brooklyn-Dodger)

Unary predicates can also be represented as binary predicates

Unary Predicate: man(Marcus)

Binary Predicate: instance(Marcus, man)

(ii) Frame

A frame is a collection of attributes and associated values that describe some entity in the world. 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. Instead build frame systems out of collections of frames that are connected to each other by a virtue of the fact that the value of an attribute of one frame may be another frame. Frame system can be used to encode knowledge and support reasoning. Each frame represents either a class or an instance.

Example of a frame

Pee-wee-Reese

instance:	:Fielder
height	:5.10
bats	:Right
batting-average	:0.309
team	:Brooklyn Dodger
uniform color	:Blue

ML-Baseball-Team

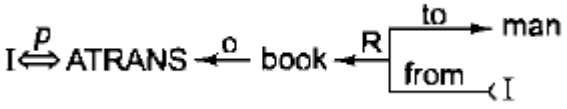
isa	:Team
cardinality	:26
team-size	:24

(iii) Conceptual Dependency

Conceptual dependency is a theory of how to represent the kind of knowledge about events that is usually contained in natural language sentence. The goal is to represent the knowledge in a way that

- Facilitates drawing inferences from the sentences
- Is independent of the language in which the sentence were originally stated

Conceptual dependency has been implemented in variety of programs that read and understand natural language text. Unlike semantic nets, which provide only a structure in to which nodes representing information at any level can be placed, conceptual dependency provides both a structure and a specific set of primitives, at a particular level of granularity, out of which representations of particular pieces of information can be constructed.



I gave the man a book

- Arrows indicate direction of dependency
- Double arrow indicate double two way link between actor and action
- P indicate past tense
- ATRANS is one of the primitive act used by the theory. It indicates transfer of possession.

- O indicates object case generation
- R indicate recipient case relation

(iv) Scripts

Script is a structure that describes a stereotyped sequence of events in a particular context. A script consists of a set of slots. Associated with each slot some information about what kind of values it may contain as well as a default value to be used if no other information is available. The important components of the scripts are

Entry conditions-Conditions that must be satisfied before the events described in a script can occur.
Result- Event happened after the script has occurred
Props- Objects involved in the event
Roles-People who are involved in the event
Track-The specific variation on a general more pattern
Scenes-The actual sequence of events that occur

Example: Restaurant
Track- Coffee Shop
Prop- Table, Menu, food, Money
Roles- S Customer
W Waiter
C cook
M Cashier
O Owner
Entry Condition
S is hungry
S has money
Results
S has less money
O has more money
S is not hungry

Scene-Eating
C ATRANS F to W
W WATRANS F to S
S INGEST F

(v) CYC

CYC is a very large knowledge base project aimed at capturing human common sense knowledge. The goal of CYC is to encode the large body of knowledge. Such a knowledge base could then be combined with specialized knowledge bases to produce systems.

CYC contains representations of events and objects. CYC is particularly concerned with the issue of scale, that is, what happens when we build knowledge bases that contain millions of objects.

Advantages of structured representation

- Retrieves the values for an attributes in a fast manner.
- Easy to describe the properties of relations.
- It is a form of object oriented programming

Unit- III

Part- A

1. **What factors determine the selection of forward or backward reasoning approach for an AI problem? (APRIL/MAY 2011)**

   Forward chaining is an example of the general concept of data-driven reasoning- ie) reasoning in which the focus of attention starts with the known data. It can be used within an agent to derive conclusions from incoming percepts, often without a query in mind.

   Backward chaining is a form of goal-directed reasoning. It is useful for answering specific questions such as "What shall I do now?" and "Where are my keys?" As the name suggests, it works backwards from the goal.

2. **Define Inference.**

   Inference is the act or process of deriving logical conclusions from premises known or assumed to be true. The conclusion drawn is also called as idiomatic.

3. **Define Forward chaining.**

   The problem solver begins with the given facts and a set of legal moves or rules for changing the state. Search proceeds by applying rules to facts to produce new facts. This process continues until (hopefully) it generates a path that satisfies the goal condition.

   Data-driven search uses knowledge and constraints found in the given data to search along lines known to be true. Use data-driven search if:

   - All or most of the data are given in the initial problem statement.
   - There are a large number of potential goals, but there are only a few ways to use the facts and the given information of a particular problem.
   - It is difficult to form a goal or hypothesis.

4. **Define Backward chaining.**

   The problem solver begins with the goal to be solved, then finds rules or moves that could be used to generate this goal and determine what conditions must be true to use them. These conditions become the new goals, subgoals, for the search. This process continues, working backward through successive subgoals, until (hopefully) a path is generated that leads back to the facts of the problem.

   Goal-driven search uses knowledge of the goal to guide the search. Use goal-driven search if;

   - A goal or hypothesis is given in the problem or can easily be formulated. (Theorem proving; medical diagnosis; mechanical diagnosis)
   - There are a large number of rules that match the facts of the problem and would thus produce an increasing number of conclusions or goals. (inefficient)
   - Problem data are not given but must be acquired by the problem solver. (e.g., medical tests determined by possible diagnosis)

5. **List various knowledge representation schemes.**

   - Rules
   - Semantic nets
   - Schemata (frames, scripts)

- Logic

6. **What is semantic network**

   Formalism for representing information about objects, people, concepts and specific relationship between them.

7. **Define inheritance**

   Inheritance mechanism allows knowledge to be stored at the highest possible level of abstraction which reduces the size of knowledge base.

8. **Mention the advantages of Production based system.**
   - Simple and easy to understand.
   - Straightforward implementation in computers possible.
   - Formal foundations for some variants.

9. **What are the disadvantages of Production Rules**
   - Simple implementation are very difficult
   - Some types of knowledge are not easily expressed in such rules.
   - Large sets of rules become difficult to understand and maintain.

10. **What are the advantages of frame based system.**
    - Fairly intuitive for many applications
      - Similar to human knowledge organization.
      - Suitable for casual knowledge.
      - Easier to understand than logic or rules.
    - Very flexible.

11. **Mention the issues of fame based system**
    - It is tempting to use fames as definitions of concepts
      - Not appropriate because there may be valid instances of a concept that do not fit the stereotype.
      - Exceptions can be used to overcome this.
    - Inheritance
      - Not all properties of a class stereotype should be propagated to subclasses.
      - Alternation of slots can have unintended consequences in subclasses.

12. **Define frame.**

    A frame is a data structure with typical knowledge about a particular object or concept. Each frame has its own name and a set of attributes associated with it.

    Ex: Name, weight, age are slots in the frame person.

13. **Give the full specification of a Bayesian network. (MAY/JUNE 2013)**

    A Bayesian network is a directed graph in which each node is annotated with quantitative probability information. The full specification is as follows:
    - A set of random variables makes up the nodes of the network. Variables may be discrete or continuous.
    - A set of directed links or arrows connects pairs of nodes. If there is an arrow from node X to node Y, X is said to be a parent of Y.
    - Each node X, has a conditional probability distribution $P(X_i|Parents(X_i))$ that quantifies the effect of the parents on the node.
    - The graph has no directed cycles (and hence a directed, acyclic graph or DAG)

14. **What do you mean by Hybrid Bayesian network? (NOV/DEC 2012)**

    A network with both discrete and continuous variables is called a hybrid Bayesian network. Two kinds of distributions are needed to specify a hybrid network:

- The conditional distribution for a continuous variable given discrete or continuous parents.
- The conditional distribution for a discrete variable given continuous parents.

15. Define Dempster-Shafer theory. (APRIL/MAY 2011)

The Dempster-Shafer theory uses interval-valued degrees of belief to represent an agent's knowledge of the probability of a proposition. The Dempster-Shafer theory is designed to deal with the distinction between uncertainty and ignorance. Rather than computing the probability of a proposition, it computes the probability that the evidence supports the proposition. This measure of belief is called a belief function, written as Bel(X).

16. Define Bayes' rule. (NOV/DEC 2012& MAY/JUNE 2013)

The simple equation of Bayes' rule underlies all modern AI systems for probabilistic inference.

The product rule can be written in two forms because of the commutativity of conjuction.

$P(a \wedge b) = P(a|b) \cdot P(b)$

$P(a \wedge b) = P(b|a) \cdot P(a)$

Equating the two right hand sides and dividing by P(a), the Bayes' rule is given as

$$P(b|a) = \frac{P(a|b)P(b)}{P(a)}$$

17. Define Certainty factors.

MYCIN represents most of its diagnostic knowledge as a set of rules. Each rule has associated with it a certainty factor, which is a measure of the extent to which the evidence that is described by the antecedent of the rule supports the conclusion is given in the rule's consequent.

18. Write the values of certainty factor

- Positive CF- evidence supports the hypothesis
- CF=1- evidence definitely proves the hypothesis
- CF=0- there is no evidence or the belief and disbelief cancel each other
- Negative CF- evidence favours negation of the hypothesis- more reason to disbelieve the hypothesis than believe it.

19. Write briefly about fuzzy set theory.

Fuzzy set theory is a means of specifying how well an object satisfies a vague description. For example, consider the proposition "He is tall." This proposition cannot be answered with an exact "true" or "false" because it is relative. That is, if the person is a three-year old and standing alongside a baby, then he is tall. But if the person is standing beside his father. Then he is not tall. Therefore, the degree of tallness needs to be considered here. Fuzzy set theory treats Tall as a fuzzy predicate and says that the truth value of Tall(Nate) is a number between 0 and 1, rather than being just true or false. The name "fuzzy set" derives from the interpretation of the predicate as implicitly defining a set of its members-a set that does not have sharp boundaries.

20. What is Noisy-OR relation? Give an example.

Uncertain relationships can often be characterized by so-called "noisy" logical relationships. The standard example is the noisy-OR relation. In propositional logic, we might say that Fever is true if and only if Cold, Flu, or Malaria is true. The noisy-OR model allows for uncertainty about the ability of each parent to cause the child to be true-

the causal relationship between parent and child may be inhibited, and so a patient could have a cold, but not exhibit a fever.

The model makes two assumptions:

- It assumes that all the possible causes are listed. (Leak node can be added that covers "miscellaneous causes.")
- It assumes that inhibition of each parent is independent of inhibition of any other parents: for example, whatever inhibits Malaria from causing a fever is independent of whatever inhibits Flu from causing a fever.

21. Give the full specification of a Bayesian network. (MAY/JUNE 2013)

A Bayesian network is a directed graph in which each node is annotated with quantitative probability information. The full specification is as follows:

- A set of random variables makes up the nodes of the network. Variables may be discrete or continuous.
- A set of directed links or arrows connects pairs of nodes. If there is an arrow from node X to node Y, X is said to be a parent of Y.
- Each node X, has a conditional probability distribution $P(X_i|Parents(X_i))$ that quantifies the effect of the parents on the node.
- The graph has no directed cycles (and hence a directed, acyclic graph or DAG)

22. What are the various kinds of knowledge?

- Declarative
  - A symbolic expression of competence
  - Declarative knowledge is abstract
  - Declarative knowledge is used to communicate and to reason
- Procedural
  - A series of steps to solve a problem
  - A compiled expression of knowledge
- Reactive
  - Stimulus- response

Part-B

1. Describe the various issues in knowledge representation.

The fundamental goal of knowledge representation is to facilitate inferencing from knowledge. The issues that arise while using knowledge representation techniques are many. Some of these are explained below.

- Any attributes of objects so basic that they appear in almost every problem domain?. If there are we need to make sure that they are handled appropriately in each of the mechanisms we propose.
- Any important relationship that exists among object attributes?
- At what level of detail should the knowledge be represented?
- How sets of objects be represented?
- Given a large amount of knowledge stored in a database, how can relevant parts be accessed?

i) Important Attributes

There are two attributes that are of very general significance

- Isa
- Instance

These attributes are important because they support property inheritance. They represent class membership and class inclusion and the class inclusion is transitive

ii) Relationships among Attributes

The attributes that are used to describe objects are themselves entities. The relationship between the Each entity have four properties independent of the specific knowledge they encode. They are:

(a) Inverses
(b) Existence in an isa hierarchy
(c) Techniques for reasoning about values
(d) Single-valued attributes

(a) Inverses:

Entities in the world are related to each other in many different ways. Relationships are attributes such as **isa, instance and team.** There are two good ways to represent the relationship.

The first is to represent both relationships in a single representation

team(Pee-Wee-Reese, Brooklyn-Dodgers) can equally easily be represent as a statement about Pee-Wee-Reese or about the Brooklyn-Dodgers.

The second approach is to use attributes that focus on a single entity but to use them in pairs, one the inverse of the other. In this approach, represent the team information with two attributes:

team = Brooklyn-Dodgers

team-members = Pee-Wee-Reese;…

This is the approach that is taken in semantic net and frame-based systems. Each time a value is added to one attribute then the corresponding value is added to the inverse.

(b) Existence in an isa hierarchy

The attribute height is the specialization of general attribute physical size which is in turn a specialization of physical attribute. The generalization specialization attributes are important because they support inheritance.

(c) Techniques for reasoning about values

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

- Information about the type of value. Eg height must be a unit of length.
- Constraints on the value. Eg age of a person cannot be greater than the age of persons parents.
- Rules for computing the value when it is needed. Eg bats attribute. These rules are called backward rules. Such rules are also called if-needed rules.
- Rules that describe actions that should be taken if a value ever becomes known. These rules are called forward rules or sometimes if-added rules.

(d) Single-valued attributes

This is about specific attribute that is guaranteed to take unique value. Eg. A baseball player can at a time have only a single height and be a member of only one team.

(iii) Choosing the granularity of representations

It deals with what level should the knowledge be represented. The primitives are

- 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, Consider the following fact

John spotted Smith

This could be represented as

Spotted(John, Smith)

Or

Spotted(agent(John), object(smith))

Such representation would make it easy to answer question such as

Who spotted Smith?

Suppose we want to know

Did John see smith?

Given only one fact, but we cannot discover the answer. So we can add another fact

Spotted(x, y)→saw(x, y)

We can now infer the answer to the question

(iv)Representing set of objects

Certain properties of objects that is true as member of a set but not as individual.

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 to attach assertion to the sets representing people, sheep and English. The reason to represent sets of object 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 explicitly with every elements of the set. This is done in different ways:

- In logical representation through the use of universal quantifier,
- In hierarchical structure where node represents 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.
- Asserting some property that holds for individual elements of the set, means, anything that is an elephant is large.

There are three ways in which sets may be represented

- Names
- Extensional definition
- Intentional definition

(v) Finding the right structures as needed

To access the right structure for describing a particular situation, it is necessary to solve all the following problems,

- How to perform an initial selection of the most appropriate structure
- How to fill in appropriate details from the current situation
- 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 structure is appropriate
- When to create and remember a new structure

iii. Selecting an Initial structure
   - Index the structure
   - Pointer to to all the structures
   - Locate one major due in the problem description

iv. Revising the choice when necessary

2. How does an inference engine work in a frame based system?

A frame is a data structure with typical knowledge about a particular object or concept. Frames, first proposed by **Marvin Minsky** in the 1970s.

Each frame has its own name and a set of **attributes** associated with it.

Frame - Person

Name
Weight
Height
Age

} attributes or slots

Each attribute or slot has a value attached to it. Frames provide a natural way for the structured and brief representation of knowledge. Frames are an application of **object-oriented programming** for expert systems.

When an object is created in an object-oriented programming language, we first assign a name to the object, and then determine a set of attributes to describe the object's characteristics, and at last write procedures to specify the object's behaviour. A knowledge engineer refers object as a **frame.**

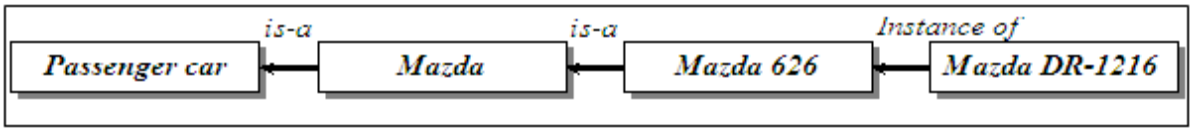
Frames as a knowledge representation technique

- The concept of a frame is defined by a collection of **slots**. Each slot describes a particular attribute or operation of the frame.
- Slots are used to store values. A slot may contain a default value or a pointer to another frame, a set of rules or procedure by which the slot value is obtained.

Typical information included in a slot

- **Frame name.**
- **Relationship of the frame to the other frames.** The frame *IBM Aptiva S35* might be a member of the class *Computer*, which in turn might belong to the class *Hardware*.
- **Slot value.** A slot value can be symbolic, numeric or Boolean. For example, the slot *Name* has symbolic values, and the slot *Age* numeric values
- **Default slot value.** The default value is taken to be true when no evidence to the contrary has been found. For example, a car frame might have four wheels and a chair frame four legs as default values in the corresponding slots.
- **Range of the slot value.** The range of the slot value determines whether a particular object complies with the stereotype requirements defined by the frame. For example, the cost of a computer might be specified between $750 and $1500.
- **Procedural information.** A slot can have a procedure attached to it, which is executed if the slot value is changed or needed.

Class inheritance in frame-based systems



CLASS:	*Passenger car*
[C]	***Engine type*** *In-line 4 cylinder:* *V6:*
[N]	***Horsepower:***
[C]	***Drivetrain type*** *Rear wheel drive:* *Front wheel drive:* *Four wheel drive:*
[C]	***Transmission type*** *5-speed manual:* *4-speed automatic:*
[N]	***Fuel consumption (mpg):***
[N]	***Seating capacity:***

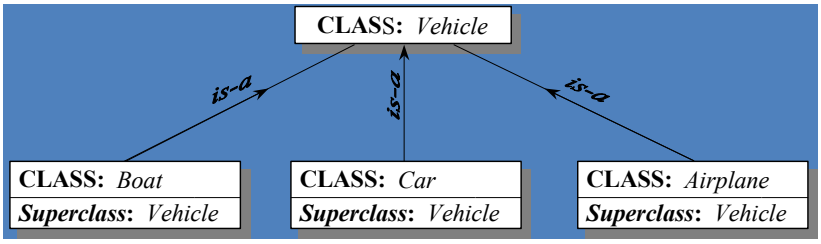
CLASS:	*Mazda*
Superclass:	*Passenger car*
[C]	***Engine type*** *In-line 4 cylinder:* *V6:*
[N]	***Horsepower:***
[C]	***Drivetrain type*** *Rear wheel drive:* *Front wheel drive:* *Four wheel drive:*
[C]	***Transmission type*** *5-speed manual:* *4-speed automatic:*
[N]	***Fuel consumption (mpg):***
[N]	***Seating capacity:***
[Str]	***Country of manufacture:*** *Japan*

CLASS:	*Mazda 626*	
Superclass:	*Mazda*	
[C]	***Engine type***	
	In-line 4 cylinder:	
	V6:	
[N]	***Horsepower:***	*125*
[C]	***Drivetrain type***	
	Rear wheel drive:	
	Front wheel drive:	
	Four wheel drive:	
[C]	***Transmission type***	
	5-speed manual:	
	4-speed automatic:	
[N]	***Fuel consumption (mpg):***	*22*
[N]	***Seating capacity:***	*5*
[Str]	***Country of manufacture:***	*Japan*
[Str]	***Model:***	
[C]	***Colour***	
	Glacier White:	
	Sage Green Metallic:	
	Slate Blue Metallic:	
	Black Onyx Clearcoat:	
[Str]	***Owner:***	

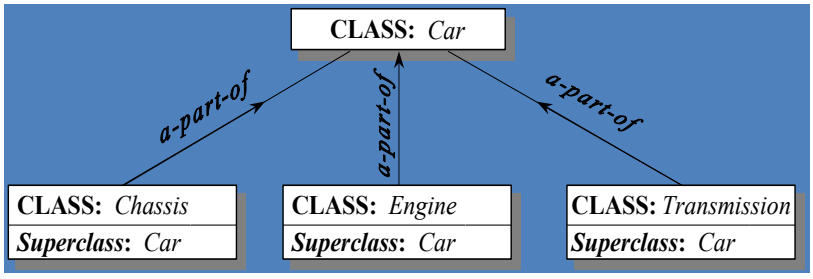
INSTANCE:	*Mazda DR-1216*	
Class:	*Mazda 626*	
[C]	***Engine type***	
	In-line 4 cylinder:	TRUE
	V6:	FALSE
[N]	***Horsepower:***	*125*
[C]	***Drivetrain type***	
	Rear wheel drive:	*FALSE*
	Front wheel drive:	TRUE
	Four wheel drive:	FALSE
[C]	***Transmission type***	
	5-speed manual:	FALSE
	4-speed automatic:	TRUE
[N]	***Fuel consumption (mpg):***	*28*
[N]	***Seating capacity:***	*5*
[Str]	***Country of manufacture:***	*Japan*
[Str]	***Model:***	*DX*
[C]	***Colour***	
	Glacier White:	FALSE
	Sage Green Metallic:	TRUE
	Slate Blue Metallic:	FALSE
	Black Onyx Clearcoat	FALSE
[Str]	***Owner:***	*Mr Black*

Relationships among objects

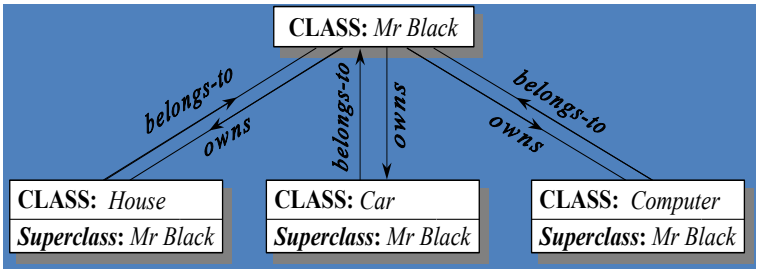
- **Generalisation** denotes *a-kind-of* or *is-a* relationship between superclass and its subclasses. Each subclass inherits all features of the superclass.



- **Aggregation** is ***a-part-of*** or ***part-whole*** relationship in which several subclasses representing *components* are associated with a superclass representing a *whole*.
  For example, an engine is *a part of* a car.



- **Association** describes some semantic relationship between different classes which are unrelated otherwise. Classes *House*, *Car* and *Computer* are mutually independent, but they are linked with the frame *Mr Black* .



- Most frame-based expert systems use two types of methods:
  - **WHEN CHANGED**
  - **WHEN NEEDED**.

A WHEN CHANGED method is executed immediately when the value of its attribute changes.
A WHEN NEEDED method is used to obtain the attribute value only when it is needed.

Interaction of frames and rules

Most frame-based expert systems allow us to use a set of rules to evaluate information contained in frames.

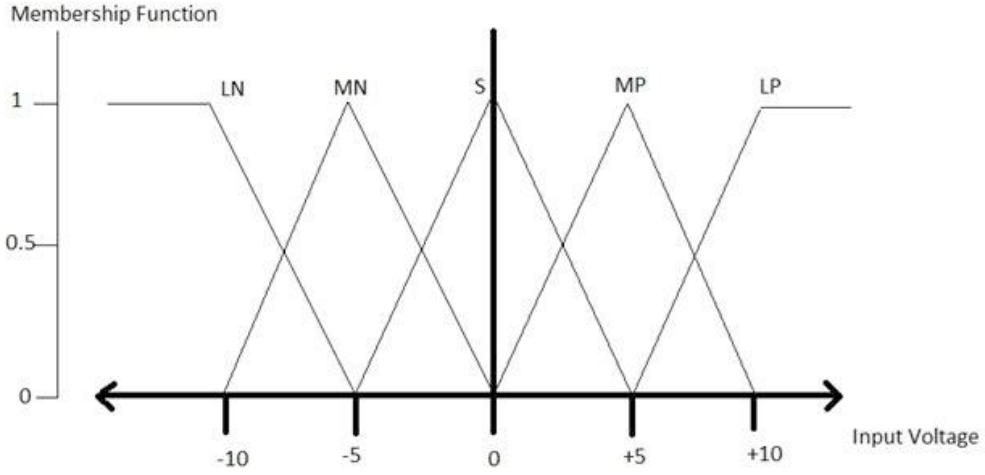
3. Explain the need of fuzzy set and fuzzy logic with example. (MAY/JUNE 2012 & NOV/DEC 2013)

Need for fuzzy logic

- Fuzzy logic is useful for commercial and practical purposes.
- It can control machines and consumer products.
- It may not give accurate reasoning, but acceptable reasoning.
- Fuzzy logic helps to deal with the uncertainty in engineering

A **membership function** for a fuzzy *set A* on the universe of discourse X is defined as μA:X→[0,1].Here, each element of *X* is mapped to a value between 0 and 1. It is called membership value or degree of membership.

All membership functions for **LP, MP, S, MN,** and **LN** are shown as below −



Here, the input to 5-level fuzzifier varies from -10 volts to +10 volts. Hence the corresponding output also changes.

Example of a Fuzzy Logic System

Let us consider an air conditioning system with 5-lvel fuzzy logic system. This system adjusts the temperature of air conditioner by comparing the room temperature and the target temperature value.

Algorithm

- Define linguistic variables and terms.
- Construct membership functions for them.
- Construct knowledge base of rules.
- Convert crisp data into fuzzy data sets using membership functions. (fuzzification)
- Evaluate rules in the rule base. (interface engine)
- Combine results from each rule. (interface engine)
- Convert output data into non-fuzzy values. (defuzzification)

Logic Development

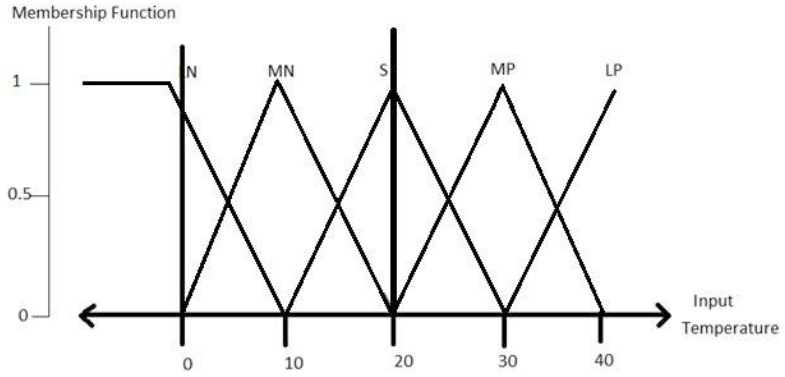
Step 1: Define linguistic variables and terms

Linguistic variables are input and output variables in the form of simple words or sentences. For room temperature, cold, warm, hot, etc., are linguistic terms.

Temperature (t) = {very-cold, cold, warm, very-warm, hot}

Step 2: Construct membership functions for them

The membership functions of temperature variable are as shown −



Step3: Construct knowledge base rules

Create a matrix of room temperature values versus target temperature values that an air conditioning system is expected to provide.

RoomTemp. /Target	Very_Cold	Cold	Warm	Very Warm	Hot
Very_Cold	No_Change	Heat	Heat	Heat	Heat
Cold	Cool	No_Change	Heat	Heat	Heat
Warm	Cool	Cool	No_Change	Heat	Heat
Very Warm	Cool	Cool	Cool	No_Change	Heat
Hot	Cool	Cool	Cool	Cool	No_Change

Build a set of rules into the knowledge base in the form of IF-THEN-ELSE structures.

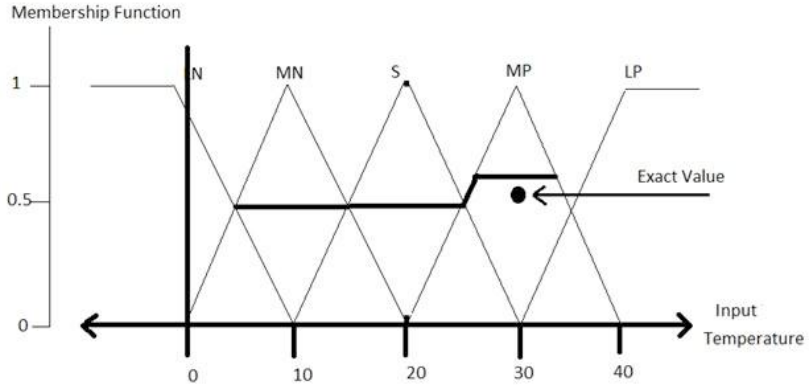
Sr. No.	Condition	Action
1	IF temperature=(Cold OR Very_Cold) AND target=Warm THEN	Heat
2	IF temperature=(Hot OR Very_Hot) AND target=Warm THEN	Cool
3	IF (temperature=Warm) AND (target=Warm) THEN	No_Change

Step 4: Obtain fuzzy value

Fuzzy set operations perform evaluation of rules. The operations used for OR and AND are Max and Min respectively. Combine all results of evaluation to form a final result. This result is a fuzzy value.

Step 5: Perform defuzzification

Defuzzification is then performed according to membership function for output variable.



4. Explain the method of performing exact inference in Bayesian networks. (NOV/DEC 2012)

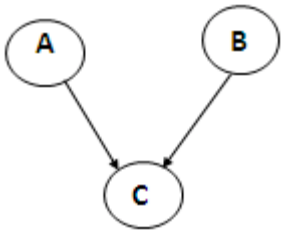
- A Bayesian network is a probabilistic graphical model that represents a set of variables and their probabilistic independencies. Otherwise known as Bayes net, Bayesian belief Network or simply Belief Networks. A Bayesian network specifies a joint distribution in a structured form. It represent dependencies and independence via a directed graph. Networks of concepts linked with conditional probabilities.

- Bayesian network consists of
  - Nodes = random variables
  - Edges = direct dependence
- Directed edges => direct dependence
- Absence of an edge => conditional independence
- Requires that graph is acyclic (no directed cycles)
- 2 components to a Bayesian network
  - The graph structure (conditional independence assumptions)
  - The numerical probabilities (for each variable given its parents)

For eg, evidence says that lab produces 98% accurate results. It means that a person X has 98% malaria or 2% of not having malaria. This factor is called uncertainty factor. This is the reason that we go for Bayesian theory. Bayesian theory is also known as probability learning.

The probabilities are numeric values between 0 and 1 that represent uncertainties.

i) Simple Bayesian network

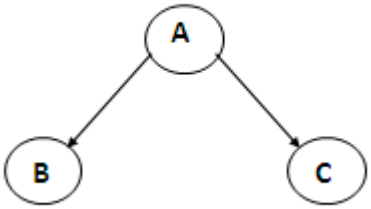


$p(A,B,C) = p(C|A,B)p(A)p(B)$

ii) 3-way Bayesian network (Marginal Independence)

$p(A,B,C) = p(A)\ p(B)\ p(C)$

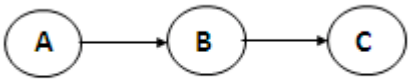
iii) 3-way Bayesian network (Conditionally independent effects)



$p(A,B,C) = p(B|A)p(C|A)p(A)$

B and C are conditionally independent Given A

iv) 3-way Bayesian network (Markov dependence)



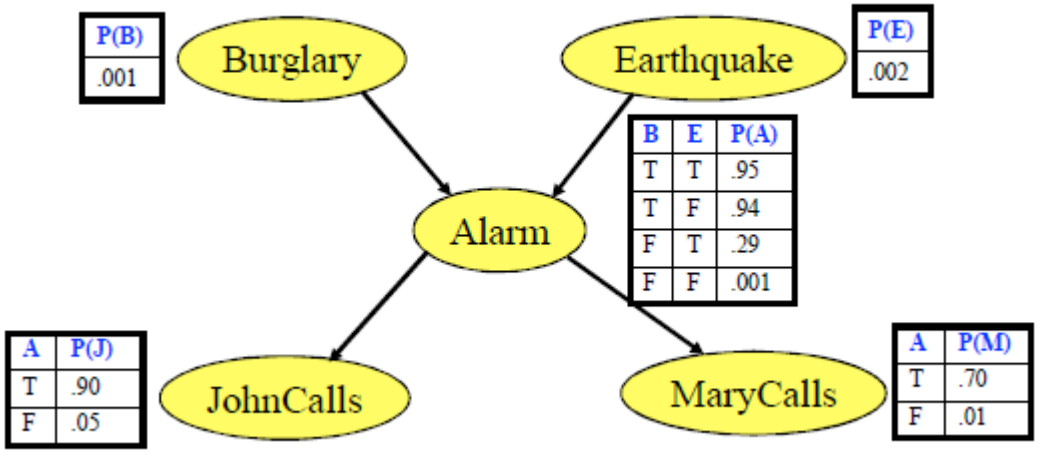
$p(A,B,C) = p(C|B)\ p(B|A)p(A)$

Problem 1

You have a new burglar alarm installed. It is reliable about detecting burglary, but responds to minor earth quakes. Two neighbors (John, Mary) promise to call you at work when they hear the alarm. John always calls when hears alarm, but confuses with phone ringing. Mary likes lod nusic and sometimes misses alarm. Find the probability of the event that the alarm has sounded but neither a burglary nor an earth quake has occurred and both Mary and John call.

Consider 5 binary variables

B=Burglary occurs at your house
E=Earth quake occurs at your home
A=Alarm goes off
J=John calls to report alarm
M=Mary calls to report the alarm



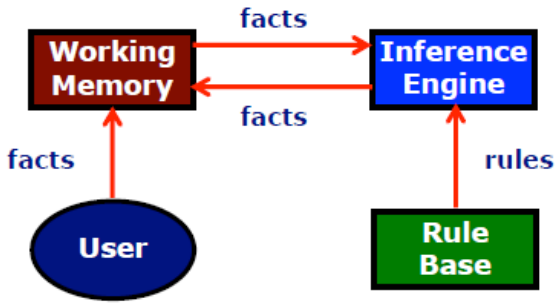
Probability of the event that the alarm has sounded but neither a burglary nor an earth quake has occurred and both Mary and John call

$$P(J,M,A, \neg E, \neg B)=P(J|A).P(M|A).P(A|\neg E, \neg B).P(\neg E).P(\neg B)$$
$$=0.90*0.70*0.001*0.99*0.998$$
$$=0.00062$$

5. Explain in detail about forward and backward chaining with suitable example.

FORWARD CHAINING

Forward chaining is also known as data driven approach. Forward chaining starts with the facts and see what rules apply.



Facts are held in the working memory (ie., contains the current state of the world). Rules represent the action to be taken when specified facts occur in the working memory. The actions involve adding or deleting facts from the working memory.

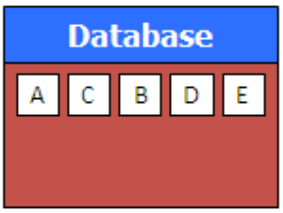
Algorithm: Forward chaining

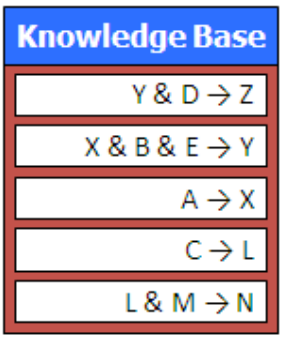
i) Collect the rules whose conditions match facts in working memory.
ii) If more than one rule matches then
(a) Use conflict resolution strategy to select the rules.
iii) Do the actions indicated by the rules. ie add facts to working memory or delete facts from working memory.
iv) Repeat these steps until the goal is reached or no condition match found.\

Example
Rule R1: IF hot and smoky THEN fire

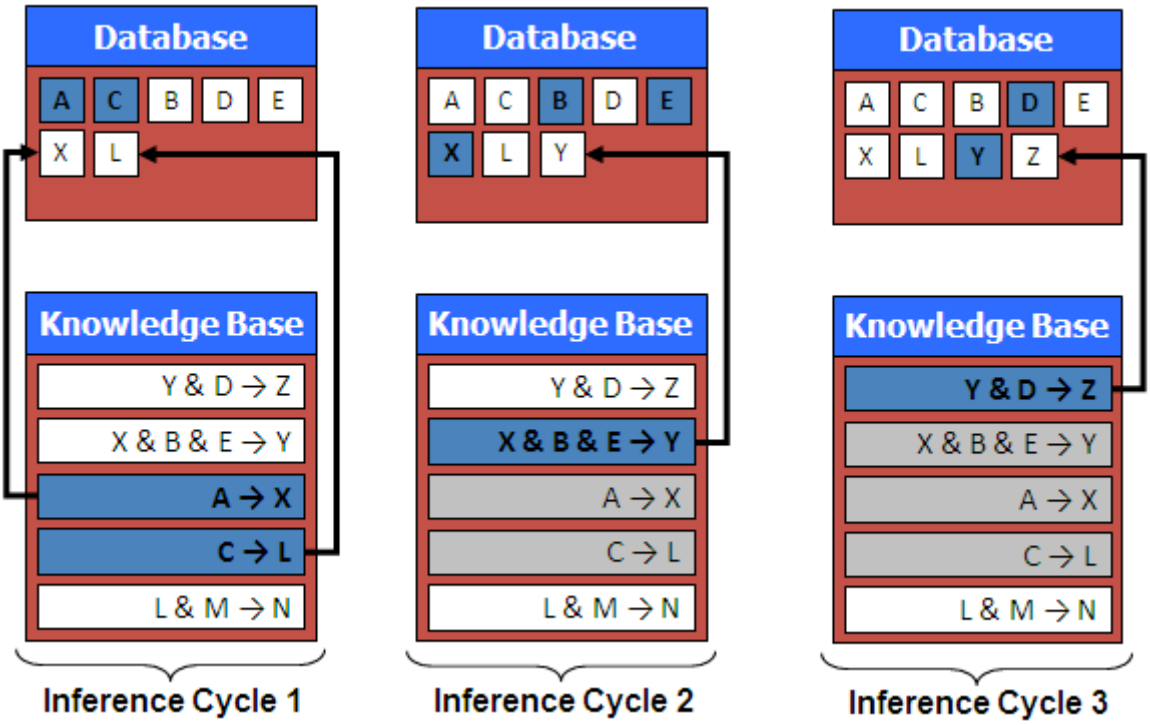
Rule R2: IF alarm_beeps THEN smoky
Rule R3: IF fire THEN switch_on_sprinklers
Fact F1: alarm_beeps (Given)
Fact F2: hot (Given)

Inference
Rule R1: IF hot and smoky THEN fire
Rule R2: IF alarm_beeps THEN smoky
Rule R3: IF fire THEN switch_on_sprinklers
Fact F1: alarm_beeps (Given)
Fact F2: hot (Given)
Fact F3: smoky (Added)
Fact F4: fire (Added)
Fact F5: switch_on_sprinklers (Added)





Suppose that we know the facts A, B, C, D, E and the rules shown in the knowledge base to the left
What facts can we infer from this?



After inferring facts X, L, Y and Z there are no more rules that can be fired.

Properties of forward chaining

- All rules which can fire do fire.
- Can be inefficient, which lead to spurious rule firing, unfocussed problem solving.
- Set of rules that can fire known as conflict set.
- Decision about which rule to fire is conflict resolution.

BACKWARD CHAINING

This is also called goal driven approach. A desired goal is placed in working memory, inference cycle attempts to find evidence to prove it.

The knowledge base (rule base) is searched for rules that might lead to goal. If condition of such rule matches fact in working memory then rule is fired and goal is proved.

Backward chaining means reasoning from goals to facts. Rules and facts are processed using backward chaining interpreter.

Algorithm: Backward Chaining

i) Prove goal G.

ii) if G is the initial fact, it is proven.

iii) Otherwise, find a rule which can be used to conclude G, and try to prove each of the rules conditions.

Example 1

Rule R1: IF hot and smoky THEN fire

Rule R2: IF alarm_beeps THEN smoky

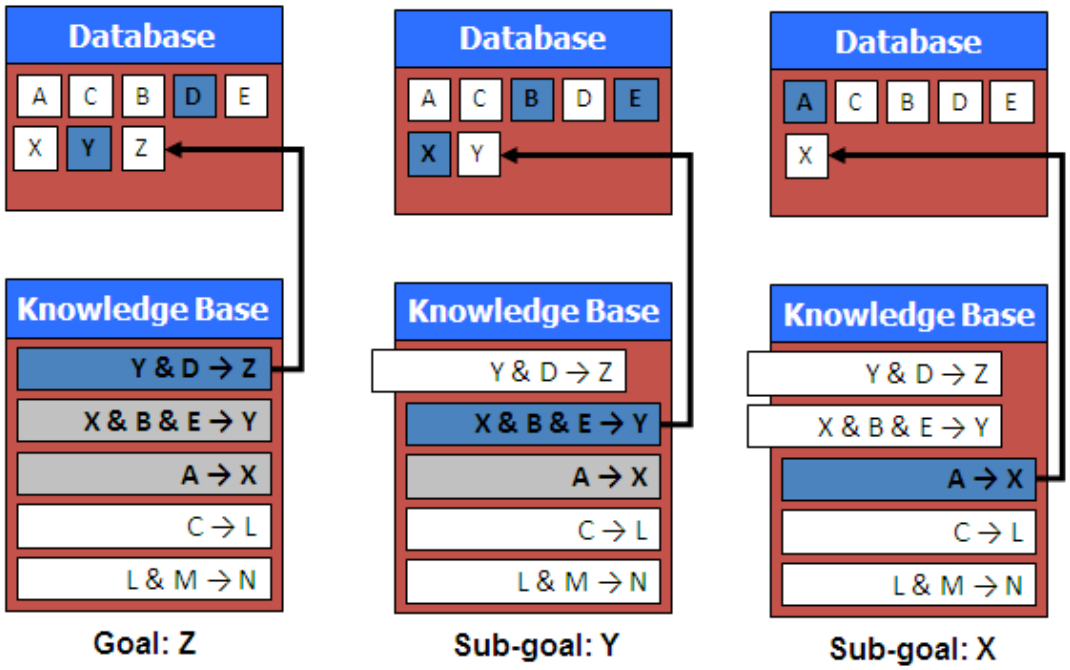
Rule R3: IF fire THEN switch_on_sprinklers

Fact F1: alarm_beeps (Given)

Fact F2: hot (Given)

Goal: Should I switch_on_sprinklers

Suppose that we know the facts A, B, C, D, E and the rules shown in the knowledge base to the left. Can we infer the fact Z?



Backward chaining inferred Z from the facts and rules that were available.

Advantages

- Most efficient to infer one particular fact.
- User may be asked to input additional facts

6. Explain knowledge representation in detail with example.

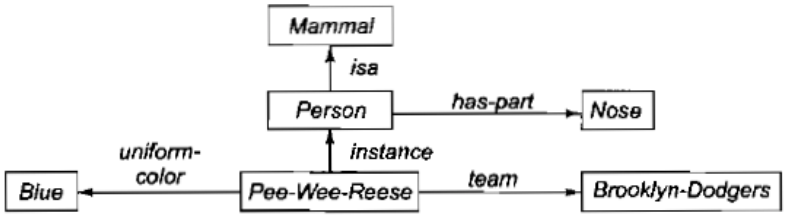
Modeling-based representations reflect the structure of the domain, and then reason based on the model.

- **Semantic Nets**
- **Frames**
- **Conceptual Dependency**
- **Scripts**
- **CYC**
- **Logic**
  - **Proposional logic**
  - **Predicate Logic**

(i) Semantic Networks

In semantic net, information is represented as a set of nodes connected to each other by a set of labelled arcs, which represent relationship among the nodes. In this network, inheritance is used to derive the additional relation.

Eg has_part(Pee-Wee-Reesse, nose)



Intersection search

To find the relationship among objects by spreading activation out from each of two nodes and seeing where the activation function met. This process is called intersection search.

Example:

What is the connection between the Brooklyn dodger and blue?

The answer is Pee-Wee-Reese

Representing non binary predicates

Semantic nets are a natural way to represent relationships that would appear as binary predicates.

instance(pee-Wee-Reese, Person)

team(Pee-Wee-Reese, Brooklyn-Dodger)

Unary predicates can also be represented as binary predicates

Unary Predicate: man(Marcus)

Binary Predicate: instance(Marcus, man)

(ii) Frame

A frame is a collection of attributes and associated values that describe some entity in the world. 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. Instead build frame systems out of collections of frames that are connected to each other by a virtue of the fact that the value of an attribute of one frame may be another frame. Frame system can be used to encode knowledge and support reasoning. Each frame represents either a class or an instance.

Example of a frame

Pee-wee-Reese

instance:	:Fielder
height	:5.10
bats	:Right
batting-average	:0.309
team	:Brooklyn Dodger
uniform color	:Blue

ML-Baseball-Team

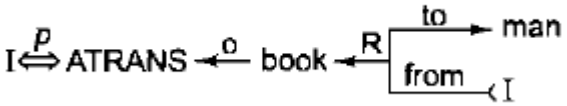
isa	:Team
cardinality	:26
team-size	:24

(iii) Conceptual Dependency

Conceptual dependency is a theory of how to represent the kind of knowledge about events that is usually contained in natural language sentence. The goal is to represent the knowledge in a way that

- Facilitates drawing inferences from the sentences
- Is independent of the language in which the sentence were originally stated

Conceptual dependency has been implemented in variety of programs that read and understand natural language text. Unlike semantic nets, which provide only a structure in to which nodes representing information at any level can be placed, conceptual dependency provides both a structure and a specific set of primitives, at a particular level of granularity, out of which representations of particular pieces of information can be constructed.



I gave the man a book

- Arrows indicate direction of dependency
- Double arrow indicate double two way link between actor and action
- P indicate past tense
- ATRANS is one of the primitive act used by the theory. It indicates transfer of possession.
- O indicates object case generation
- R indicate recipient case relation

(iv) Scripts

Script is a structure that describes a stereotyped sequence of events in a particular context. A script consists of a set of slots. Associated with each slot some information about what kind of values it may contain as well as a default value to be used if no other information is available. The important components of the scripts are

Entry conditions-Conditions that must be satisfied before the events described in a script can occur.

Result- Event happened after the script has occurred

Props- Objects involved in the event

Roles-People who are involved in the event

Track-The specific variation on a general more pattern

Scenes-The actual sequence of events that occur

Example: Restaurant

Track- Coffee Shop

Prop- Table, Menu, food, Money

Roles- S Customer
W Waiter
C cook
M Cashier
O Owner

Entry Condition
S is hungry
S has money

Results
S has less money
O has more money
S is not hungry

Scene-Eating
C ATRANS F to W
W WATRANS F to S
S INGEST F

(v) CYC

CYC is a very large knowledge base project aimed at capturing human common sense knowledge. The goal of CYC is to encode the large body of knowledge. Such a knowledge base could then be combined with specialized knowledge bases to produce systems.

CYC contains representations of events and objects. CYC is particularly concerned with the issue of scale, that is, what happens when we build knowledge bases that contain millions of objects.

(vi) Propositional Logic

A *proposition* is a declarative statement that's either TRUE or FALSE (but not both).

Propositions	Not Propositions
3 + 2 = 32	Bring me coffee!
CS173 is Bryan's favorite class.	CS173 is her favorite class.
Every cow has 4 legs.	3 + 2
There is other life in the universe.	Do you like Cake?

- **The symbols of the language:**
  - Propositional symbols (Prop): A, B, C,…
  - Connectives:
    - ∧ and
    - ∨ or
    - ¬ not
    - → implies
    - ↔ equivalent to
    - ⊗ xor (different than)
    - ⊥, > False, True
  - Parenthesis :(,).

(vii) Predicate Logic

FOL or predicate logic is a generalization of propositional logic that allows us to express and infer arguments in infinite models like

v. All men are mortal
vi. Some birds cannot fly

Predicates are like functions except that their return type is true or false.

A predicate with no variable is a proposition.

Syntax of First-Order Logic

- Constants KingJohn, 2, …
- Predicates/Relation Brother, >, …
- Functions Sqrt, LeftArmOf, …
- Variables x, y, a, b, …
- Connectives ∧ ∨ ¬ ⇒ ⇔
- Equality =
- Quantifiers $ "

Components of First-Order Logic

- **Term**
  - Constant, e.g. Red
  - Function of constant, e.g. Color(Block1)
- **Atomic Sentence**
  - Predicate relating objects (no variable)

- Brother (John, Richard)
- Married (Mother(John), Father(John))

- **Complex Sentences**
  - Atomic sentences + logical connectives
    - Brother (John, Richard) ∧¬Brother (John, Father(John))
- **Quantifiers**
  - Each quantifier defines a variable for the duration of the following expression, and indicates the truth of the expression…
- **Universal quantifier "for all" ∀**
  - The expression is true for every possible value of the variable
- **Existential quantifier "there exists" ∃**
  - The expression is true for at least one value of the variable

Examples

1. Some dogs bark.

∃x dog(x) ∧ bark(x)

2. All dogs have four legs.

∀x(dog(x)→have_four_legs(x))

(or)

∀x(dog(x)→legs(x,4))

3. All barking dogs are irritating

∀x(dog(x) ∧barking(x)→irritating(x))

4. No dogs purr.

¬∃x (dog(x) ∧ purr(x))

7. Explain Rule based system with example.

Production systems are also known as **rule based system**.

A system whose knowledge base is represented as a set of rules and facts is called rule based system. A rule based system consists of a collection of IF-THEN rules, a collection of facts, and some interpreter controlling the application of the rules given the facts.

Production rules

- IF-THEN expressions
- **IF** some condition(s) exists **THEN** perform some action(s)

Condition-action pair

- **Condition:** pattern that determines when a rule may be applied to problem instance
- **Action:** defines associated problem solving step

Antecedent – Consequent

- IF <antecedent> THEN <consequent>

When the antecedent part is NULL, the rule becomes fact.

Rule can have multiple antecedents

Conjunction AND

IF <antecedent$_0$> AND <antecedent$_1$> … AND <antecedent$_n$>
THEN <consequent>

Disjunction OR

IF <antecedent$_0$> OR <antecedent$_1$> … OR <antecedent$_n$>
THEN <consequent>

Combination of both

IF <antecedent0> AND <antecedent1> … AND <antecedentn>

OR <antecedent$_0$> OR <antecedent$_1$> … OR <antecedent$_n$>
THEN <consequent>

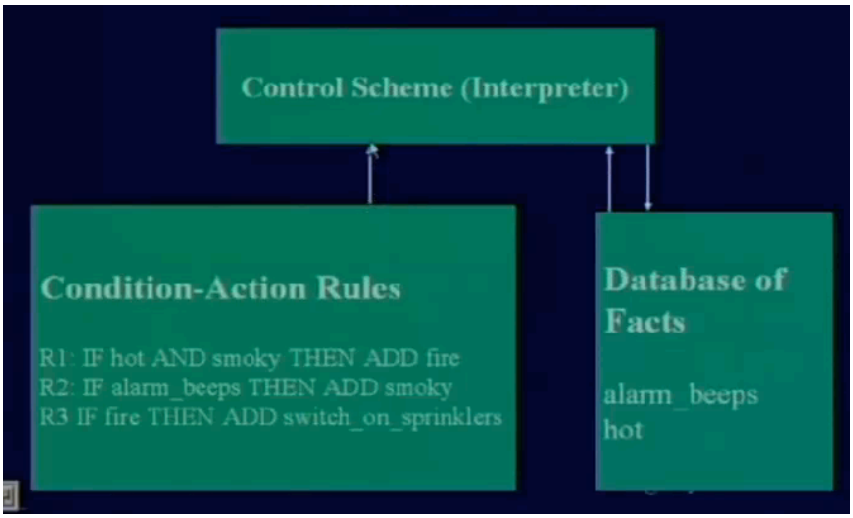
Consequents can also have multiple clauses

IF <antecedent> THEN <consequent$_0$>, <consequent$_1$>,
…<consequent$_{n-1}$>, <consequent$_n$>

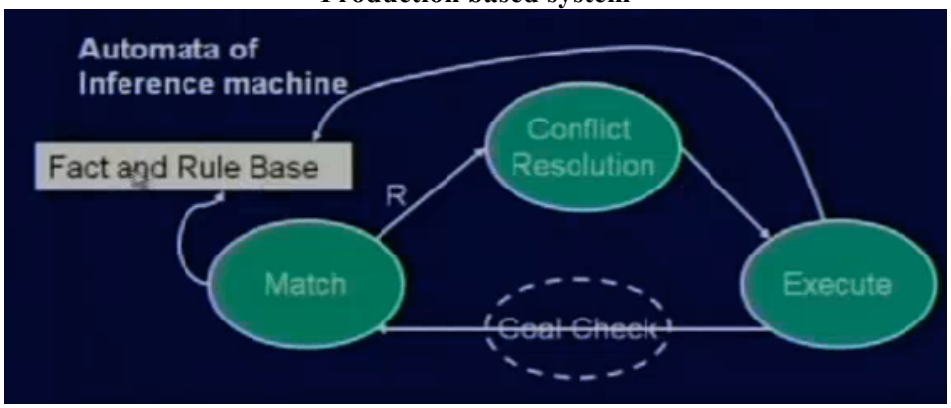
Triggered and fired rules

A rule is **triggered** when all the antecedents evaluate to true.

A rule is fired when the action stated in the consequent part or the inference related to the consequent part is inferred or taken.



Production based system



Inference machine is a machine that implements strategies to utilize the knowledge base and derive new conclusions from it.

Match

Inference machine is comparing the fact base and the rule base and find some rules are matching. These set of rules are passed to the next phase, conflict resolution.

Conflict resolution phase

- *Rule* 1:IF the 'traffic light' is green THEN the action is go
- *Rule* 2:IF the 'traffic light' is red THEN the action is stop
- *Rule* 3:IF the traffic light' is red THEN the action is go

We have two rules, *Rule* 2 and *Rule* 3, with the same IF part. Thus both of them can be set to fire when the condition part is satisfied. These rules represent a conflict set. The inference engine must determine which rule to fire from such a set. A method for choosing a rule to fire when more than one rule can be fired in a given cycle is called **conflict resolution**.

The conflict resolution mechanisms are

- Rule Ordering
- Recency
- Specificity
- Refraction

i) Recency

- Rules which use more recent facts are preferred.
- Working memory elements are time tagged indicating at what cycle each fact was added to working memory.
- Focuses on single line of reasoning

ii) Specificity

- Rules which have greater number of conditions are therefore more difficult to satisfy, are preferred to more general rules with fewer conditions.
- More specific rules are better because they take more of the data in to account.

iii) Refraction

- A rule should not be allowed to fire more than once on the same data.
- The executed rules are discarded from the conflict set.
- Prevents undesired loops

iv) Rule ordering

- Choose the first rule in the text ordered top to bottom

Alternative to conflict resolution

Meta rules reason about which rules should be considered for firing. They direct reasoning rather than actually performing reasoning. Meta knowledge is knowledge about knowledge to guide search.

If conflict set contains any rule (c,a) such that c="animal is mammal" THEN fire(c,a). This example says meta knowledge encodes knowledge about how to guide search for solution.

Execute phase

The selected rule is fired in this phase. This phase also checks whether the goal is reached. After every execution new fact will be added to the fact base.

Limitations of Rule-Based Representations

- Can be difficult to create
  - the "knowledge engineering" problem
- Can be difficult to maintain
  - in large rule-bases, adding a rule can cause many unforeseen interactions and effects => difficult to debug
- Many types of knowledge are not easily represented by rules
  - uncertain knowledge: "if it is cold it will probably rain"
  - information which changes over time
  - procedural information (e.g. a sequence of tests to diagnose a disease)

8. Explain Dempster - Shafer theory with example.

This means that it is possible to believe that something could be both true and false to some degree

Dempster-Shafer theory is an approach to combining evidence. Dempster (1967) developed means for combining degrees of belief derived from independent items of evidence.

Each fact has a degree of support, between 0 and 1:

- 0 No support for the fact
- 1 full support for the fact

• Differs from Bayesian approach in that:

- Belief in a fact and its negation need not sum to 1.
- Both values can be 0 (meaning no evidence for or against the fact)

Set of possible conclusions:

$\Theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$

– Θ is the set of possible conclusions to be drawn
– Each θ_i is mutually exclusive: at most one has to be true.
– Θ is Exhaustive: At least one θ_i has to be true.

Mass function m(A):

(where A is a member of the power set)= proportion of all evidence that supports this element of the power set.

Each m(A) is between 0 and 1.

- All m(A) sum to 1.
- m(Ø) is 0 - at least one must be true.

Belief in A:

The belief in an element A of the Power set is the sum of the masses of elements which are subsets of A (including A itself).

A={q1, q2, q3}

Bel(A) = m(q1)+m(q2)+m(q3) + m({q1, q2})+m({q2, q3})+m({q1, q3})+m({q1, q2, q3})

Plausibility of A: pl(A)

The plausibility of an element A, pl(A), is the sum of all the masses of the sets that intersect with the set A:

pl({B,J}) = m(B)+m(J)+m(B,J)+m(B,S)+m(J,S)+m(B,J,S)

Problem 1

- 4 people (B, J, S and K) are locked in a room when the lights go out.
- When the lights come on, K is dead, stabbed with a knife.
- Not suicide (stabbed in the back)
- No-one entered the room.
- Assume only one killer.

• Θ= { B, J, S}

• P(Θ) = (Ø, {B}, {J}, {S}, {B,J}, {B,S}, {J,S}, {B,J,S})

Mass function m(A):

• Detectives, after reviewing the crime-scene, assign mass probabilities to various elements of the power set:

Event	Mass
No-one is guilty	0
B is guilty	0.1
J is guilty	0.2
S is guilty	0.1
either B or J is guilty	0.1
either B or S is guilty	0.1
either S or J is guilty	0.3
One of the 3 is guilty	0.1

A	{B}	{J}	{S}	{B,J}	{B,S}	{J,S}	{B,J,S}
m(A)	0.1	0.2	0.1	0.1	0.1	0.3	0.1
bel(A)	0.1	0.2	0.1	0.4	0.3	0.6	1.0
pl(A)	0.4	0.7	0.6	0.9	0.8	0.9	1.0

The certainty associated with a given subset A is defined by the belief interval: [bel(A) pl(A)].

From this we observed that J is a killer.

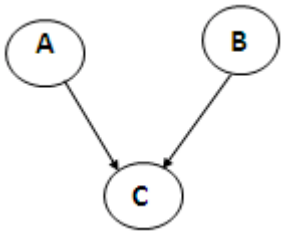
9. Describe a method for constructing Bayesian networks. (8) (MAY/JUNE 2013)

- A Bayesian network is a probabilistic graphical model that represents a set of variables and their probabilistic independencies. Otherwise known as Bayes net, Bayesian belief Network or simply Belief Networks. A Bayesian network specifies a joint distribution in a structured form. It represent dependencies and independence via a directed graph. Networks of concepts linked with conditional probabilities.
- Bayesian network consists of
  - Nodes = random variables
  - Edges = direct dependence
- Directed edges => direct dependence
- Absence of an edge => conditional independence
- Requires that graph is acyclic (no directed cycles)
- 2 components to a Bayesian network
  - The graph structure (conditional independence assumptions)
  - The numerical probabilities (for each variable given its parents)

For eg, evidence says that lab produces 98% accurate results. It means that a person X has 98% malaria or 2% of not having malaria. This factor is called uncertainty factor. This is the reason that we go for Bayesian theory. Bayesian theory is also known as probability learning.

The probabilities are numeric values between 0 and 1 that represent uncertainties.

i) Simple Bayesian network

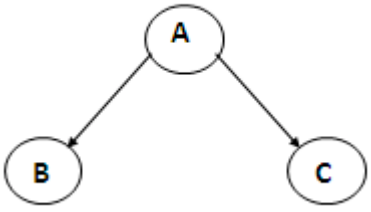


$$p(A,B,C) = p(C|A,B)p(A)p(B)$$

ii) 3-way Bayesian network (Marginal Independence)

$$p(A,B,C) = p(A)\ p(B)\ p(C)$$

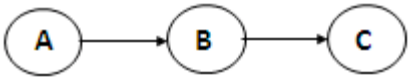
iii) 3-way Bayesian network (Conditionally independent effects)



$$p(A,B,C) = p(B|A)p(C|A)p(A)$$

B and C are conditionally independent Given A

iv) 3-way Bayesian network (Markov dependence)



$$p(A,B,C) = p(C|B)\ p(B|A)p(A)$$

10. Write notes on:

i) Fuzzy reasoning

Fuzzy Logic (FL) is a method of reasoning that resembles human reasoning. The approach of FL imitates the way of decision making in humans that involves all intermediate possibilities between digital values YES and NO. In fuzzy logic, the degree of truth is between 0 and 1.

Example: William is smart (0.8 truth)

The fuzzy logic works on the levels of possibilities of input to achieve the definite output.

Fuzzy logic is useful for commercial and practical purposes.

- It can control machines and consumer products.
- It may not give accurate reasoning, but acceptable reasoning.
- Fuzzy logic helps to deal with the uncertainty in engineering.

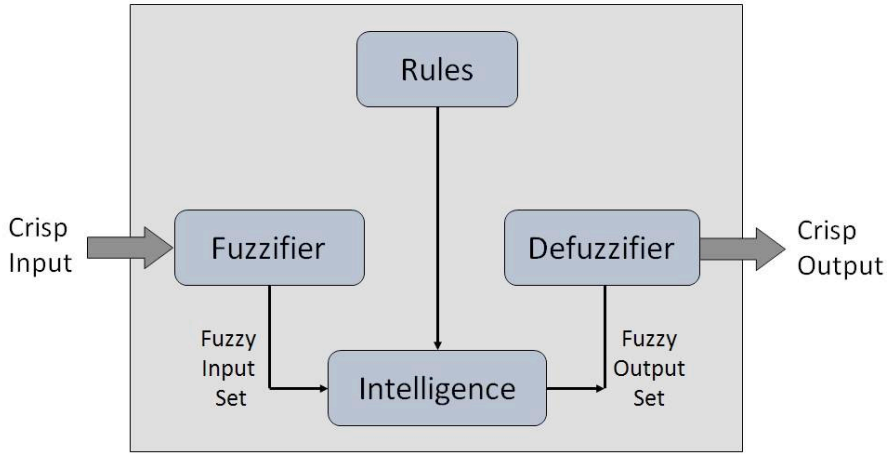
Fuzzy Logic Systems Architecture

It has four main parts as shown −

- **Fuzzification Module** − It transforms the system inputs, which are crisp numbers, into fuzzy sets. It splits the input signal into five steps such as −

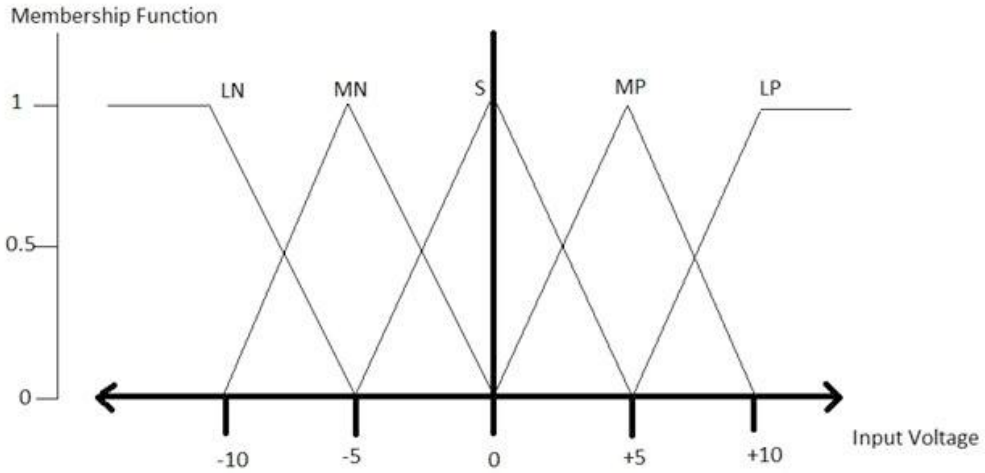
LP	x is Large Positive
MP	x is Medium Positive
S	x is Small
MN	x is Medium Negative
LN	x is Large Negative

- **Knowledge Base** − It stores IF-THEN rules provided by experts.
- **Inference Engine** − It simulates the human reasoning process by making fuzzy inference on the inputs and IF-THEN rules.
- **Defuzzification Module** − It transforms the fuzzy set obtained by the inference engine into a crisp value.



A **membership function** for a fuzzy *set A* on the universe of discourse X is defined as μA:X→[0,1].Here, each element of *X* is mapped to a value between 0 and 1. It is called membership value or degree of membership.

All membership functions for **LP, MP, S, MN,** and **LN** are shown as below −



Here, the input to 5-level fuzzifier varies from -10 volts to +10 volts. Hence the corresponding output also changes.

Example of a Fuzzy Logic System

Let us consider an air conditioning system with 5-lvel fuzzy logic system. This system adjusts the temperature of air conditioner by comparing the room temperature and the target temperature value.

Algorithm

- Define linguistic variables and terms.
- Construct membership functions for them.
- Construct knowledge base of rules.
- Convert crisp data into fuzzy data sets using membership functions. (fuzzification)
- Evaluate rules in the rule base. (interface engine)
- Combine results from each rule. (interface engine)
- Convert output data into non-fuzzy values. (defuzzification)

Logic Development

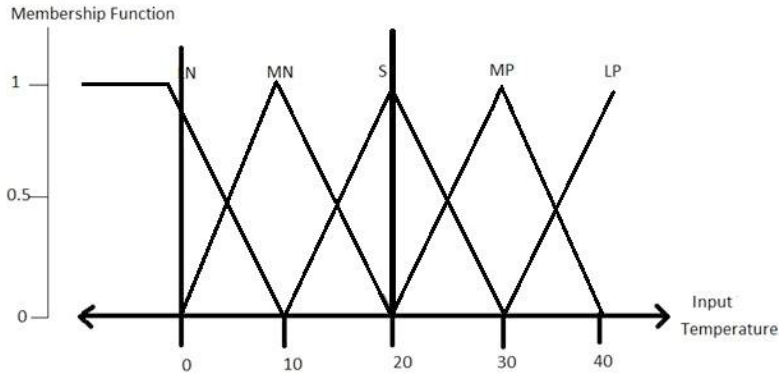
Step 1: Define linguistic variables and terms

Linguistic variables are input and output variables in the form of simple words or sentences. For room temperature, cold, warm, hot, etc., are linguistic terms.

Temperature (t) = {very-cold, cold, warm, very-warm, hot}

Step 2: Construct membership functions for them

The membership functions of temperature variable are as shown −



Step3: Construct knowledge base rules

Create a matrix of room temperature values versus target temperature values that an air conditioning system is expected to provide.

RoomTemp. /Target	Very_Cold	Cold	Warm	Very Warm	Hot
Very_Cold	No_Change	Heat	Heat	Heat	Heat
Cold	Cool	No_Change	Heat	Heat	Heat
Warm	Cool	Cool	No_Change	Heat	Heat
Very Warm	Cool	Cool	Cool	No_Change	Heat
Hot	Cool	Cool	Cool	Cool	No_Change

Build a set of rules into the knowledge base in the form of IF-THEN-ELSE structures.

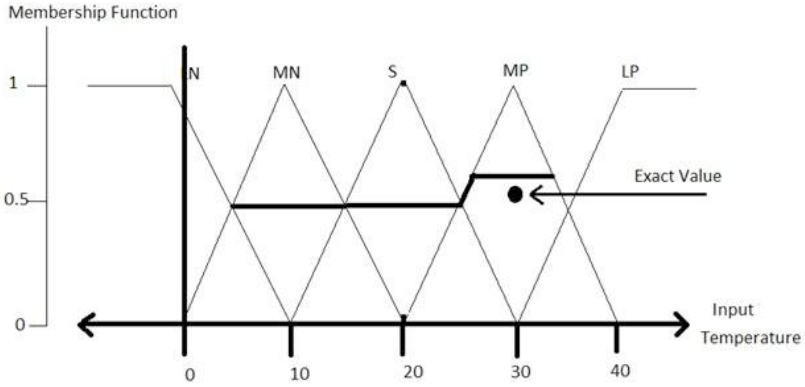
Sr. No.	Condition	Action
1	IF temperature=(Cold OR Very_Cold) AND target=Warm THEN	Heat
2	IF temperature=(Hot OR Very_Hot) AND target=Warm THEN	Cool
3	IF (temperature=Warm) AND (target=Warm) THEN	No_Change

Step 4: Obtain fuzzy value

Fuzzy set operations perform evaluation of rules. The operations used for OR and AND are Max and Min respectively. Combine all results of evaluation to form a final result. This result is a fuzzy value.

Step 5: Perform defuzzification

Defuzzification is then performed according to membership function for output variable.



Application Areas of Fuzzy Logic

Automotive Systems

- Automatic Gearboxes
- Four-Wheel Steering
- Vehicle environment control

Consumer Electronic Goods

- Hi-Fi Systems
- Photocopiers
- Still and Video Cameras
- Television

Domestic Goods

- Microwave Ovens
- Refrigerators
- Toasters
- Vacuum Cleaners
- Washing Machines

Environment Control

- Air Conditioners/Dryers/Heaters
- Humidifiers

Advantages of Fuzzy Logic System

- Mathematical concepts within fuzzy reasoning are very simple.
- Able to modify Fuzzy Logic System by just adding or deleting rules due to flexibility of fuzzy logic.
- Fuzzy logic Systems can take imprecise, distorted, noisy input information.
- FLSs are easy to construct and understand.
- Fuzzy logic is a solution to complex problems in all fields of life, including medicine, as it resembles human reasoning and decision making.

Disadvantages of Fuzzy Logic System

- There is no systematic approach to fuzzy system designing.
- They are understandable only when simple.
- They are suitable for the problems which do not need high accuracy.

ii) Bayesian probability

It is the basis of uncertain reasoning where the results are unpredictable.

Bayes Rule

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

P(h)- prior probability of hypothesis h

P(D)prior probability of data D, evident
P(h|D)-posterior probability
P(D|h)- likelihood of D given h

Axioms of probability

1. All probabilities are between 0 and 1 ie0≤P(A) ≤1
2. P(True)=1 and P(false)=0
3. P(A∨B)=P(A)+P(B)-P(A∧B)

iii) Certainty factors

A certainty factor (CF) is a numerical value that expresses a degree of subjective belief that a particular item is true. The item may be a fact or a rule. When probabilities are used attention must be paid to the underlying assumptions and probability distributions in order to show validity. Bayes' rule can be used to combine probability measures.

Suppose that a certainty is defined to be a real number between -1.0 and +1.0, where 1.0 represents complete certainty that an item is true and -1.0 represents complete certainty that an item is false. Here a CF of 0.0 indicates that no information is available about either the truth or the falsity of an item. Hence positive values indicate a degree of belief or evidence that an item is true, and negative values indicate the opposite belief. Moreover it is common to select a positive number that represents a minimum threshold of belief in the truth of an item. For example, 0.2 is a commonly chosen threshold value

Certainty Factors and Facts and Rules

CFs with facts

padre(John, Mary) with CF .90

CFs with rules

(if (sneezes X) then (has_cold X)) with CF 0.7

– where the CF measures our belief in the conclusion given the premise is observed.

CFs are calculated using two other measures:

1. MB(H, E) – Measure of Belief: value between 0 and 1 representing the degree to which belief in the hypothesis H is supported by observing evidence E.

$$MB(H,E) = \begin{cases} 1 & \text{si } p(H) = 1 \\ \frac{p(H|E) - p(H)}{1 - p(H)} & \text{si } p(H) < 1 \end{cases}$$

It is intended to capture the degree to which the evidence increases probability: p(H|E)-p(H) in proportion to the maximum possible increase in probability: 1-p(H)

2. MD(H, E) – Measure of Disbelief: value between 0 and 1 representing the degree to which disbelief in the hypothesis H is supported by observing evidence E.

$$MD(H,E) = \begin{cases} 1 & \text{si } p(H) = 0 \\ \frac{p(H) - p(H|E)}{p(H)} & \text{si } p(H) > 0 \end{cases}$$

CF is calculated in terms of the difference between MB and MD:

$$CF(H,E) = MB(H,E) - MD(H,E)$$

Combining Certainty Factors

• Multiple sources of evidence produce CFs for the same fact.
• For instance, two rules may provide evidence for the same conclusion:
– if P1 Then C CF=0.8

– if P2 Then C CF=0.7
• We need to know how to combine the CFs in such cases
• If two rules both support a hypothesis, then that should increase our belief in the hypothesis.

To combine certainty factors, use an incremental approach as with Bayes

$$CF(H, E \wedge E') = \begin{cases} X + Y(1 - X) & \text{if } X, Y > 0 \\ X + Y(1 + X) & \text{if } X, Y < 0 \\ \frac{X+Y}{1-min(|X|,|Y|)} & \text{if } sign(X) \neq sign(Y) \end{cases}$$

Rules with uncertain evidence
When a rule has a single premise, the certainty of the conclusion is the PRODUCT of the certainty of the premise multiplied by the certainty of the rule:

CF(C) = CF(P) * CF(R1)

Rules with uncertain evidence: negative evidence
A rule is only applicable if you believe the premise to be true

CF(C) = CF(P) * CF(RULE) if CF(P)>0
= 0 otherwise

Rules with uncertain evidence: more than one premise
If a rule has more than one premise:
IF P1&P2&P3 THEN C
• We find the CF of the set of premises – WHICH is just the MIN
CFPs = MIN(CF(P1), CF(P2), CF(P3))
CF(C) = CFPs * CF(RULE) if CFPs>0
= 0 otherwise

Unit- IV
Part- A

1. **Define planning.**
   Planning is arranging a sequence of actions to achieve a goal.
   Ex: Navigation, Language processing.
2. **What is Non Linear Plan.**
   - Non linear planning uses goal set instead of goal stack.
   - Include in the search space all possible subgoal orderings
     - Handles goal interactions by interleaving.
3. **What are the advantages and disadvantages of Non linear plan**
   Advantages:
   - Non linear planning is sound, complete.
   - Optimal with respect to plan length(depending on search strategy employed)

   Disadvantages:
   - Larger search space, since possible goal orderings may have to be considered.
   - Somewhat more complex algorithms.
4. **Define STRIPES.**
   STRIPES (Stanford Research Institute Problem Solver) is an automated planner. This language is the base for most of the languages for expressing automated planning problem instances. A STRIPS instance is composed of:
   An Initial state, the specification of goal states, A set of actions (each action includes- Preconditions and post conditions).

5. **Mention the components of Planning system.**
   - Choosing Rules to Apply
   - Applying Rules
   - Detecting a Solution
   - Detecting Dead Ends
   - Repairing a almost correct solution
6. **What is Goal Stack?**

   One of the earliest technique to be developed for solving compound goals that may interact was the use of a goal stack. This was the approach used by STRIPS. In this method, the problem solver makes use of a single stack that contains both goals and operators that have been proposed to satisfy those goals.the problem solver also relies on a database that describes as PRECONDITION,ADD, and DELETE lists.
7. **List out the various planning techniques. (MAY/JUNE 2014)**

   The various planning techniques are:
   - Planning with state-space search
     - → Forward state-space search
     - → Backward state-space search
   - Partial order planning
     - → Partial-order planning with unbound variables
   - Planning with propositional logic
   - Hierarchical task network planning
   - Planning and acting in nondeterministic domains
     - → Sensorless planning
     - → Conditional planning
     - → Execution monitoring and replanning
     - → Continuous planning
8. **What are the basic operations in the Blocks world?**

   stack(X,Y): Put block X on block Y
   unstack(X,Y): remove block X from block Y
   pickup(X): pickup block X
   putdown(X): put block X on the table
9. **Distinguish between problem solving and planning. (NOV/DEC 2012)**

   Problem solving is a mental process which is concluding part of a larger problem process that includes problem finding and problem shaping where problem is defined as state of desire for reaching a definite goal from a present condition that either is not directly moving toward the goal or stepping towards the goal. Planning systems are problem solving algorithms that operate on explicit propositional representations of states and actions. These representations make possible the derivation of effective heuristics and development of powerful and flexible algorithms for solving problems.
10. **What is a consistent plan? (MAY/JUNE 2013)**

    A consistent plan is one in which there are no cycles in the ordering constraints and no conflicts with causal links. A consistent plan with no open preconditions is a solution.
11. **Define contingency plan. (MAY/JUNE 2012 & MAY/JUNE 2014)**

    Conditional planning is also known as contingency planning, this approach deals with bounded indeterminacy by constructing a conditional plan with different branches for the different contingencies that could arise.

12. What is learning?

An agent tries to improve its behaviour through observation, reasoning, or reflection.

- Learning from experience
- Forecasting
- Theories

13. Explain the concept of learning from example. (APRIL/MAY 2011)

The idea behind learning is that percepts should be used not only for acting but also for improving the agent's ability to act in the future.

14. Define Machine learning.

A computer program is said to learn from experience E with respect to some class of tasks T and performance P, if its performance at the tasks improves with the experiences.

15. Explain Machine learning with an example.

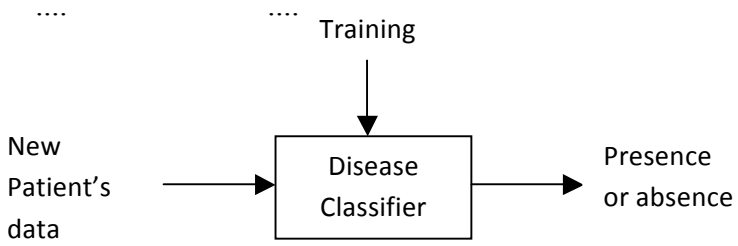
- Learns from past experience.
- Improves the performance of intelligent programs.

Ex: Disease diagnosis

Database of medical records:

Patient 1's data- Absence

Patient 2's data- Presence



16. Mention the advantages of Machine Learning.

- Alleviate knowledge acquisition bottleneck
  - Does not require knowledge engineers.
  - Scalable in constructing knowledge base.
- Adaptive
  - Adaptive to the changing conditions.
  - Easy in migrating to new domains.

17. What is generalization

The number of distinct objects that might potentially be stored can be very large. To keep the number of stored objects down to a manageable level, some kind of generalization is necessary.

18. Give examples for planning problem.

- Route search
  - Find a route from source to destination.
- Military operations
  - Develop an air campaign.
- Game playing
  - Plan the behaviour of computer controlled player.

- Resources control
    - Plan the stops of several of elevators in a skyscraper building.

19. What are the components of a Learning agent

- Learning element
- Performance element
- Critic
- Problem generator

20. What are the various forms of learning?

- Supervised learning
- Unsupervised learning
- Reinforcement learning

21. Define concept- learning.

The idea of producing a classification problem that can evolve its own class definitions is appealing. This task of constructing class definition is called concept-learning or induction.

Part-B

1. Explain STRIPS mechanism with example.

Planning refers to the process of computing several steps of a problem solving procedure before executing any of them.

STRIPS

Its name is derived from STanford Research Institute Problem Solver.

- An automated planner.
- Planning system for a robotics project: SHAKEY.
- Developed by Richard Fikes and Nils Nilsson in 1971 at SRI International.
- Classical Planning System.
- Knowledge Representation: First Order Logic.
- Algorithm: Forward chaining on rules.

STRIPS-Like Planning Formulation

- A finite, nonempty set of *instances*.
- A finite, nonempty set of *predicates*, which are binary-valued (partial) functions of one of more instances. Each application of a predicate to a specific set of instances is called a *positive literal*. A logically negated positive literal is called a *negative literal*.
- A finite, nonempty set of *operators*, each of which has:

  1) *preconditions*, which are positive or negative literals that must hold for the operator to apply, and

  2) *effects*, which are positive or negative literals that are the result of applying the operator.

- An *initial set* which is expressed as a set of *positive literals*. Negative literals are implied. For any positive literal that does not appear in , its corresponding negative literal is assumed to hold initially.
- A *goal set* which is expressed as a set of both *positive* and *negative literals*.

STRIPS Planner

- STRIPS maintains **two additional data structures**:
  - **State List** - all currently true predicates.
  - **Goal Stack** - a push down stack of goals to be solved, with current goal on top of stack. *f*
- If the current goal is not satisfied by present state,
  - Find goal in the add list of an operator, and push operator and preconditions list on stack. (=Sub goals).
- When a current goal is satisfied, OP it from stack. *f*
- When an operator is on top of the stack,
  - record the application of that operator – update the plan sequence and
  - use the operator's add and delete lists to update the current state.

Reasoning Loop

- If the top item on the goal stack is:
  - empty (the goal stack is empty), return the **actions** executed
    - they form the plan to achieve the goal
  - a **goal**, and it is **satisfied** in the current state, remove it from the stack (no replacement necessary)
  - a complex goal, break it into sub goals, placing all sub goals on the goal stack (the original goal is pushed down into the goal stack)
  - a predicate, find an action that will make it true, then place that action (with variables bound appropriately) and its preconditions on the goal stack (preconditions first)
  - an action and its preconditions are satisfied, perform the action, updating the world state using the delete and add lists of the action (if the pre-conditions are not satisfied, add them to the goal list without removing the action).
  - Add this action to the partial plan.

STRIPS Limitation

- Expressive power of description language for the operators:
  - Operator changes exactly, what is specified in its add list and delete list.
- Problems with the strategy: Sussman anomaly

Example: Blocks World Problem

The world consists of:

- **A flat surface** such as a table top
- An adequate **set of identical blocks** which are **identified by letters**.
- The blocks can be **stacked** one on one to form towers of apparently unlimited height.

- Sufficiently simple and well behaved.
- Easily understood.
- Good sample environment to study planning.
  - Problems can be broken into nearly distinct sub problems.
  - We can show how partial solutions need to be combined to form a realistic complete solution.

The stacking is achieved using a robot arm which has fundamental operations and states which can be assessed using logic and combined using logical operations. The robot can hold one block at a time

and only one block can be moved at a time. Any number of blocks can be on the table. The actions it can perform include

- *stack(X,Y)*: put block X on block Y. The arm must already be holding X and the surface of Y must be clear.
- *unstack(X,Y)*: remove block X from block Y. The arm must be empty and block X must have no blocks on top of it.
- *pickup(X)*: pickup block X from the table. The arm must be empty and there must be nothing on top of X.
- *putdown(X)*: put block X on the table. The arm must have been holding block X. *f*

The **stack action** occurs when the robot arm places the object it is holding [X] on top of another object [Y].

- **FORM:** STACK(X,Y) *f*
- **PRE:** CLEAR(Y) ∧ HOLDING(X) □
- **ADD:** ARMEMPTY() ∧ ON(X,Y) ∧ CLEAR(X) □
- **DEL:** CLEAR(Y) ∧ HOLDING(X) □
- **CONSTRAINTS:** (X ≠ Y), X ≠ TABLE, Y ≠ TABLE

The **unstack action** occurs when the robot arm picks up an object X from on top of another object Y.

- **FORM:** UNSTACK(X,Y) □
- **PRE:** ON(X,Y) ∧ CLEAR(X) ∧ ARMEMPTY() □
- **ADD:** HOLDING(X) ∧ CLEAR(Y) □
- **DEL:** ON(X,Y) ∧ CLEAR(X) ∧ ARMEMPTY() □
- **CONSTRAINTS:** X ≠ Y, X ≠ TABLE, Y ≠ TABLE

The **pickup action** occurs when the arm picks up an object (block) X from the table. □

- **FORM:** PICKUP(X) □
- **PRE:** ONTABLE(X) ∧ CLEAR(X) ∧ ARMEMPTY() □
- **ADD:** HOLDING(X) □
- **CONSTRAINTS:** X ≠ TABLE

The **putdown action** occurs when the arm places the object X onto the table. □

- **FORM:** PUTDOWN(X) □
- **PRE:** HOLDING(X) □
- **ADD:** ONTABLE(X) ∧ ARMEMPTY()∧ CLEAR(X) □
- **DEL:** HOLDING(X) □
- **Constraints:** *X ≠ Table*

To do these operations, we need to use the following predicates:

- ON(X,Y) – Block x is on block Y
- ONTABLE(X) – Block X is on table
- CLEAR(X) – there is nothing on top of block X
- HOLDING(X) – The arm is holding block X.
- ARMEMPTY – The arm is holding nothing

2. Explain the various components of a Planning system.

In problem solving systems, it is necessary to perform each of the following functions:

- Choose the best rule to apply next based on the best available heuristic information
- Apply the chosen rule to compute the new problem state that arises from its application.
- Detect when a solution has been found
- Detect dead ends so that they can be abandoned and the system's effort directed in more fruitful directions.
- Detect when almost correct solution has been found and employ special techniques to make it totally correct.

i) Choosing rule to apply

The most widely used technique for selecting appropriate rules to apply is first to isolate a set of differences between the desired goal state and the current state and then to identify those rules that are relevant to reducing those differences. If several rules are found, a variety of other heuristic information can be exploited to choose among them.

ii) Applying Rules

To handle complex domains, STRIPS problem solver is used. In this approach each operation is described using three lists

- Precondition List – a list of facts which must be true for action to be executed.
- DELETE list - a list of facts that are no longer true after action is performed; *f*
- ADD list - a list of facts made true by executing the action.

Basic operations

- *stack(X,Y)*: put block X on block Y
- *unstack(X,Y)*: remove block X from block Y
- *pickup(X)*: pickup block X from the table
- *putdown(X)*: put block X on the table *f*

The **stack action** occurs when the robot arm places the object it is holding [X] on top of another object [Y].

- **FORM:** STACK(X,Y) *f*
- **PRE:** CLEAR(Y) ∧ HOLDING(X) □
- **ADD:** ARMEMPTY() ∧ ON(X,Y) ∧ CLEAR(X) □
- **DEL:** CLEAR(Y) ∧ HOLDING(X) □
- **CONSTRAINTS:** (X ≠ Y), X ≠ TABLE, Y ≠ TABLE

The **unstack action** occurs when the robot arm picks up an object X from on top of another object Y.

- **FORM:** UNSTACK(X,Y) □
- **PRE:** ON(X,Y) ∧ CLEAR(X) ∧ ARMEMPTY() □
- **ADD:** HOLDING(X) ∧ CLEAR(Y) □
- **DEL:** ON(X,Y) ∧ CLEAR(X) ∧ ARMEMPTY() □
- **CONSTRAINTS:** X ≠ Y, X ≠ TABLE, Y ≠ TABLE

The **pickup action** occurs when the arm picks up an object (block) X from the table. □

- **FORM:** PICKUP(X) □
- **PRE:** ONTABLE(X) ∧ CLEAR(X) ∧ ARMEMPTY() □
- **ADD:** HOLDING(X) □
- **CONSTRAINTS:** X ≠ TABLE

The **putdown action** occurs when the arm places the object X onto the table. □

- **FORM:** PUTDOWN(X) □
- **PRE:** HOLDING(X) □
- **ADD:** ONTABLE(X) ∧ ARMEMPTY()∧ CLEAR(X) □
- **DEL:** HOLDING(X) □
- **Constraints:** $X \neq Table$

iii) Detecting a Solution

A planning system has succeeded in finding a solution to a problem when it has found a sequence of operators that transform the initial problem state in to goal state. In simple problem solving systems, a straightforward match of the state descriptions has done.

iv) Detecting dead ends

As a planning system is searching for a sequence of operators to solve a particular problem, it must be able to detect when it is exploring a path that can never lead to a solution.

If the search process is reasoning forward from initial state, it can prune any a path that leads to a state from which the goal state cannot be reached.

If the search process is reasoning backward from the goal state, it can also terminate a path either because it is sure that the initial state cannot be reached or because little progress is being made. In reasoning backward, each goal is decomposed in to sub goals. Each of them in turn may lead to a set of additional sub goals. Sometimes it is easy to detect that there is no way that all sub goals in a given set can be satisfied at once.

v) Repairing an almost correct solution

The problem is decomposed completely, proceed to solve sub problems separately and check that when solutions are combined, they do in fact yield a solution to the original problem. If they do not throw out the solution, look for another one.

A slightly better approach to look at the situation that results when the sequence of operations corresponding to the proposed solution is executed and to compare the situation to the desired goal. In most cases, the difference between the two will be smaller than the difference between initial state and goal. Now the problem solving can be called again and asked to find a way of eliminating this new difference. The first solution can then be combined with this second one to form a solution to the original problem.

An even better way to patch up an almost correct solution is to appeal to specific knowledge about what went wrong and then apply a direct path.

A still better way to patch up incomplete solutions is not really to patch them up at all, but rather to leave them incompletely specified until the last possible moment. Then when as much information as possible is available, complete specification in such a way that no conflicts arise. This approach can be thought of as a least commitment strategy. It can be applied in variety of ways. One is to defer deciding on the order in which operations will be performed.

3. What do you mean by plan generation? Explain the concept of planning with the Block world example.

Planning refers to the process of computing several steps of a problem solving procedure before executing any of them.

STRIPS

- Its name is derived from STanford Research Institute Problem Solver.
- An automated planner.
- Planning system for a robotics project: SHAKEY.
- Developed by Richard Fikes and Nils Nilsson in 1971 at SRI International.
- Classical Planning System.
- Knowledge Representation: First Order Logic.
- Algorithm: Forward chaining on rules.

STRIPS-Like Planning Formulation

- A finite, nonempty set of *instances*.
- A finite, nonempty set of *predicates*, which are binary-valued (partial) functions of one of more instances. Each application of a predicate to a specific set of instances is called a *positive literal*. A logically negated positive literal is called a *negative literal*.
- A finite, nonempty set of *operators*, each of which has:

  1) *preconditions*, which are positive or negative literals that must hold for the operator to apply, and

  2) *effects*, which are positive or negative literals that are the result of applying the operator.

- An *initial set* which is expressed as a set of *positive literals*. Negative literals are implied. For any positive literal that does not appear in , its corresponding negative literal is assumed to hold initially.
- A *goal set* which is expressed as a set of both *positive* and *negative literals*.

BLOCK WORLD PROBLEM

- The world consists of:
  - **A flat surface** such as a table top
  - An adequate **set of identical blocks** which are **identified by letters**.
  - The blocks can be **stacked** one on one to form towers of apparently unlimited height.

Why Use the Blocks World as an Example?

- Sufficiently simple and well behaved.
- Easily understood.
- Good sample environment to study planning.
  - Problems can be broken into nearly distinct sub problems.
  - We can show how partial solutions need to be combined to form a realistic complete solution.

The stacking is achieved using a robot arm which has fundamental operations and states which can be assessed using logic and combined using logical operations. The robot can hold one block at a time and only one block can be moved at a time. Any number of blocks can be on the table. The actions it can perform include

- *stack(X,Y)*: put block X on block Y. The arm must already be holding X and the surface of Y must be clear.
- *unstack(X,Y)*: remove block X from block Y. The arm must be empty and block X must have no blocks on top of it.
- *pickup(X)*: pickup block X from the table. The arm must be empty and there must be nothing on top of X.
- *putdown(X)*: put block X on the table. The arm must have been holding block X. *f*

The **stack action** occurs when the robot arm places the object it is holding [X] on top of another object [Y].

- **FORM:** STACK(X,Y) □
- **PRE:** CLEAR(Y) ∧ HOLDING(X) □
- **ADD:** ARMEMPTY() ∧ ON(X,Y) ∧ CLEAR(X) □
- **DEL:** CLEAR(Y) ∧ HOLDING(X) □
- **CONSTRAINTS:** (X ≠ Y), X ≠ TABLE, Y ≠ TABLE

The **unstack action** occurs when the robot arm picks up an object X from on top of another object Y.

- **FORM:** UNSTACK(X,Y) □
- **PRE:** ON(X,Y) ∧ CLEAR(X) ∧ ARMEMPTY() □
- **ADD:** HOLDING(X) ∧ CLEAR(Y) □
- **DEL:** ON(X,Y) ∧ CLEAR(X) ∧ ARMEMPTY() □
- **CONSTRAINTS:** X ≠ Y, X ≠ TABLE, Y ≠ TABLE

The **pickup action** occurs when the arm picks up an object (block) X from the table. □

- **FORM:** PICKUP(X) □
- **PRE:** ONTABLE(X) ∧ CLEAR(X) ∧ ARMEMPTY() □
- **ADD:** HOLDING(X) □
- **CONSTRAINTS:** X ≠ TABLE

The **putdown action** occurs when the arm places the object X onto the table. □

- **FORM:** PUTDOWN(X) □
- **PRE:** HOLDING(X) □
- **ADD:** ONTABLE(X) ∧ ARMEMPTY()∧ CLEAR(X) □
- **DEL:** HOLDING(X) □
- **Constraints:** *X ≠ Table*

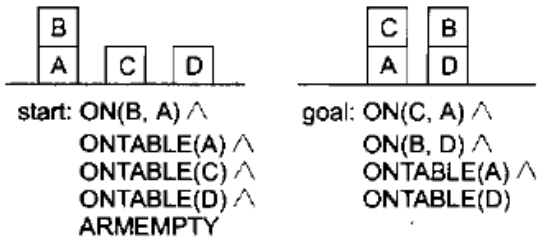
To do these operations, we need to use the following predicates:

- ON(X,Y) – Block x is on block Y
- ONTABLE(X) – Block X is on table
- CLEAR(X) – there is nothing on top of block X
- HOLDING(X) – The arm is holding block X.
- ARMEMPTY – The arm is holding nothing

4. Discuss simple planning using a Goal Stack.

One of the earliest techniques to be developed for solving compound goals that may interact was the use of a goal stack.

This approach was used by STRIPS. A single stack is maintained which contains both goals and operators.



When we begin solving this problem, the goal stack is simply

ON(C,A) ∧ON(B,D) ∧ONTABLE(A) ∧ONTABLE(D)

Separate this problem in to sub problems, one for each component of the original goal. Two of the sub-problems ONTABLE(A) and ONTABLE(D) are already true in the initial state. OTAD is the abbreviation for ONTABLE(A) ∧ ONTABLE(D)

ON(C,A)
ON(B,D)
ON(C,A) ∧ON(B,D) ∧OTAD

At each succeeding step of the problem solving process, the top goal on the stack will be pursued. When a sequence of operators that satisfies is found, that sequence is applied to the state description, yielding a new description. Next, the goal that is then at the top of the stack is explored and an attempt is made to satisfy it, starting from the situation that was produced as a result of satisfying the first goal. This process continues until the goal is empty.

Then as one last check, the original goal is compared to the final state derived from the application of the chosen operators. If any components of the goal are not satisfied in that state, then those unsolved parts of the goal are reinserted on to the stack and the process resumed.

ON(C,A) is replaced with the operator STACK(C,A).

STACK(C,A)
ON(B,D)
ON(C,A) ∧ON(B,D) ∧OTAD

The preconditions of STACK(C,A) is pushed inside the stack.

CLEAR(A)
HOLDING(C)
CLEAR(A) ∧HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) ∧ON(B,D) ∧OTAD

CLEAR(A) is not true. To make CLEAR(A) true, B must be un-stacked from A. This is done by the operator UNSTACK(B,A).

UNSTACK(B,A)
HOLDING(C)
CLEAR(A) ∧HOLDING(C) STACK(C,A)
ON(B,D)
ON(C,A) ∧ON(B,D) ∧OTAD

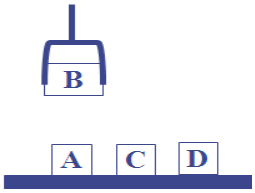
The preconditions of UNSTACK(B,A) are pushed inside the goal stack.

ON(B,A)
CLEAR(B)
ARMEMPTY
ON(B,A) ∧CLEAR(B) ∧ARMEMPTY
UNSTACK(B,A)
HOLDING(C)

CLEAR(A) ∧HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) ∧ON(B,D) ∧OTAD

Here, ON(B,A) is true, CLEAR(B) is true, ARMEMPTY is true. So, they are popped out from the goal stack. Thus, all the preconditions of UNSTACK(B,A) is true. So, UNSTACK(B,A) is placed in the solution list.

Solution List={ UNSTACK(B,A) }



HOLDING(C) is not true because the arm is holding B.

HOLDING(C)
CLEAR(A) ∧HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) ∧ON(B,D) ∧OTAD

Thus, HOLDING(C) is replaced with the operator PICKUP(C)

PICKUP(C)
CLEAR(A) ∧HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) ∧ON(B,D) ∧OTAD

The preconditions for PICKUP(C) are pushed inside the goal stack.

ONTABLE(C)
CLEAR(C)
ARMEMPTY
ONTABLE(C) ∧CLEAR(C) ∧ARMEMPTY
PICKUP(C)
CLEAR(A) ∧HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) ∧ON(B,D) ∧OTAD

Here, ONTABLE(C) is true, CLEAR(C) is true. So ONTABLE(C) and CLEAR(C) are popped out from the goal stack. ARMEMPTY is not true because the arm is holding C. Thus, make the arm empty by stacking B on D. This done by the operator STACK(B.D), which replaces ARMEMPTY.

STACK(B,D)
ONTABLE(C) ∧CLEAR(C) ∧ARMEMPTY

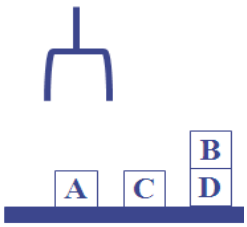
PICKUP(C)
CLEAR(A) ∧HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) ∧ON(B,D) ∧OTAD

The preconditions of STACK(B,D) is pushed inside the goal stack.

CLEAR(D)
HOLDING(B)
CLEAR(D) ∧HOLDING(B)
STACK(B,D)
ONTABLE(C) ∧CLEAR(C) ∧ARMEMPTY
PICKUP(C)
CLEAR(A) ∧HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) ∧ON(B,D) ∧OTAD

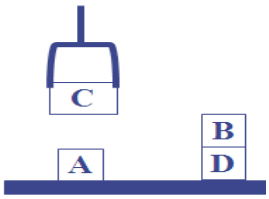
Here, CLEAR(D) and HOLDING(B) are true. This means that the preconditions of STACK(B,D) is true. Thus, CLEAR(D) and HOLDING(B) are popped out and STACK(B,D) is placed in the solution list.

Solution List={ UNSTACK(B,A), STACK(B,D)}



ONTABLE(C) ∧CLEAR(C) ∧ARMEMPTY
PICKUP(C)
CLEAR(A) ∧HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) ∧ON(B,D) ∧OTAD

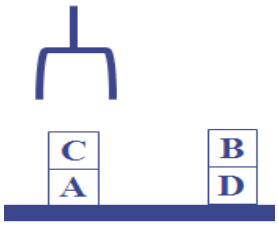
Now, ONTABLE(C), CLEAR(C) and ARMEMPTY are true. So they are popped out and PICKUP(C) is placed on the solution list.
Solution List={ UNSTACK(B,A), STACK(B,D), PICKUP(C)}



CLEAR(A) ∧HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) ∧ON(B,D) ∧OTAD

Here, CLEAR(A) and HOLDING(C) are true. Thus, CLEAR(A) and HOLDING(C) are popped out from the goal stack. STACK(C,A) is placed in the solution list.

Solution List={ UNSTACK(B,A), STACK(B,D), PICKUP(C), STACK(C,A)}



Thus, the goal is reached.

5. Write briefly the concept of Rote Learning and learning by taking advice.

Learning process is the basis of knowledge acquisition process. Knowledge acquisition is the expanding the capabilities of a system or improving its performance at some specified task. The acquired knowledge may consist of various facts, rules, concepts, procedures, heuristics, formulas, relationships or any other useful information.

Rote Learning

It is the simplest form of learning. It requires the least amount of inference and is accomplished by simply copying the knowledge in the same form that it will be used directly into the knowledge base. It includes learning my imitation, simple memorization and learning by being performed.

For example we may use this type of learning when we memorize multiplication tables. In this method we store the previous computed values, for which we do not have for recomputed them later. Also we can say rote learning is one type of existing or base learning.

For example, in our childhood, we have the knowledge that “sun rises in the east”. So in our later stage of learning we can easily memorize the thing. Hence in this context, a system may simply memorize previous solutions and recall them when confronted with the same problem. Generally access of stored value must be faster than it would be to re-compute. Methods like hashing, indexing and sorting can be employed to enable this.

One drawback of rote learning is it is not very effective in a rapidly changing environment. If the environment does change then we must detect and record exactly what has changed. Also this technique must not decrease the efficiency of the system.

Learning by Taking Advice

In this process we can learn through taking advice from others. The idea of advice taking learning was proposed in early 1958 by McCarthy. In our daily life, this learning process is quite common. Right from our parents, relatives to our teachers, when we start our educational life, we take various advices from others. We know the computer programs are written by programmers. When a programmer writes a computer program he or she gives many instructions to computer to follow, the same way a teacher gives his/her advice to his students. The computer follows the instructions given by the programmer. Hence, a kind of learning takes place when computer runs a particular program by taking advice from the creator of the program.

Mostow describes a program called FOO, which accepts advice for playing hearts, a card game. A human user first translates the advice from English in to a representation that FOO can understand.

Example: (avoid(take-points me)(trick))

6. Explain Machine learning.

Machine learning is the systematic study of algorithms and systems that improve their knowledge or performance (learn a model for accomplishing a task) with experience (from available data /examples)

Examples:

- Given an URL decide whether it is a Sports website or not
- Given that a buyer is buying a book at online store, suggest some related products for that buyer
- Given an ultrasound image of abdomen scan of a pregnant lady, predict the weight of the baby
- Given a CT scan image set, decide whether there is stenosis or not
- Given marks of all the students in a class, assign relative grades based on statistical distribution
- Given a mail received, check whether it is a SPAM
- Given a speech recording, identify the emotion of the speaker
- Given a DNA sequence, predict the promoter regions in that sequence

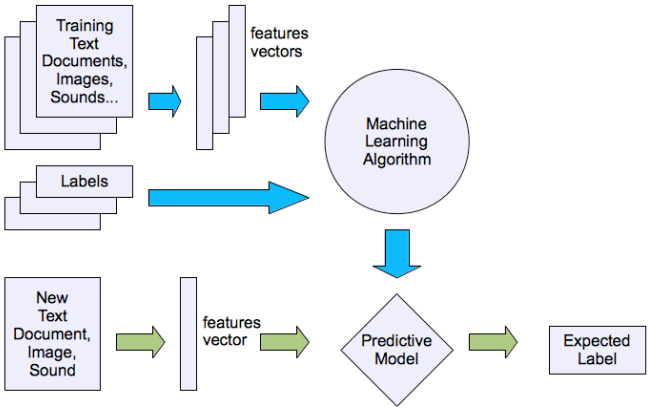
These are some examples of “Intelligent tasks” — tasks that are “easy” for humans but “extremely difficult” for a machine to achieve Artificial Intelligence is about building systems that can efficiently perform such “intelligent tasks”

One of the important aspects that enable humans to perform such intelligent tasks is their ability to learn from experiences (either supervised or unsupervised)

Machine learning tasks are typically classified into three broad categories, depending on the nature of the learning "signal" or "feedback" available to a learning system.

(i) Supervised learning

The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs.



- **Prediction**
- **Classification (discrete labels),**
- **Regression (real values)**

Prediction

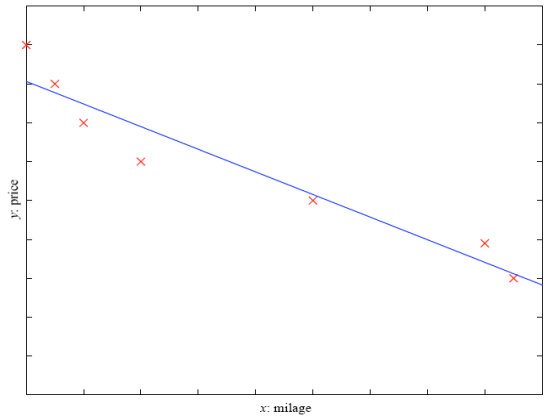
Example: Price of a used car

x : car attributes

y : price

$y = g\ (x \mid \theta\ )$

θ parameters



Classification

Example 1

Suppose you have a basket and it is filled with different kinds of fruits. Your task is to arrange them as groups. For understanding let me clear the names of the fruits in our basket.

You already learn from your previous work about the physical characters of fruits. So arranging the same type of fruits at one place is easy now. Your previous work is called as training data in data mining. You already learn the things from your train data; this is because of response variable. Response variable means just a decision variable.

No.	SIZE	COLOR	SHAPE	FRUIT NAME
1	Big	Red	Rounded shape with a depression at the	Apple
2	Small	Red	Heart-shaped to nearly globular	Cherry
3	Big	Green	Long curving cylinder	Banana
4	Small	Green	Round to oval, Bunch shape Cylindrical	Grape

Suppose you have taken a new fruit from the basket then you will see the size, color and shape of that particular fruit. If size is Big , color is Red , shape is rounded shape with a depression at the top, you will conform the fruit name as apple and you will put in apple group.

If you learn the thing before from training data and then applying that knowledge to the test data (for new fruit), this type of learning is called as Supervised Learning.

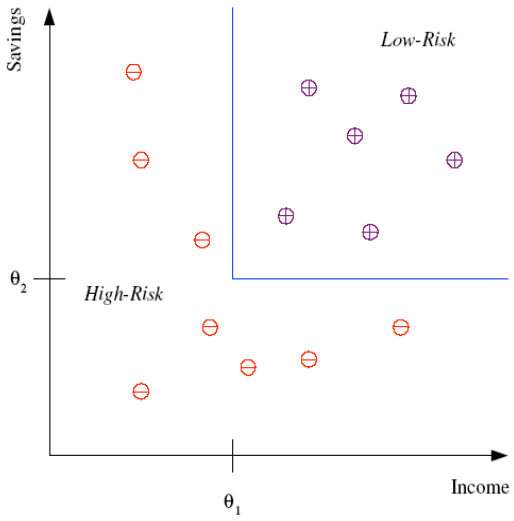
Example 2

Credit scoring

Differentiating between low-risk and high-risk customers from their *income* and *savings*

Discriminant: IF *income* > θ_1 AND *savings* > θ_2

THEN low-risk ELSE high-risk



Regression

Given example pairs of heights and weights of a set of people, find a model to predict the weight of a person from her height

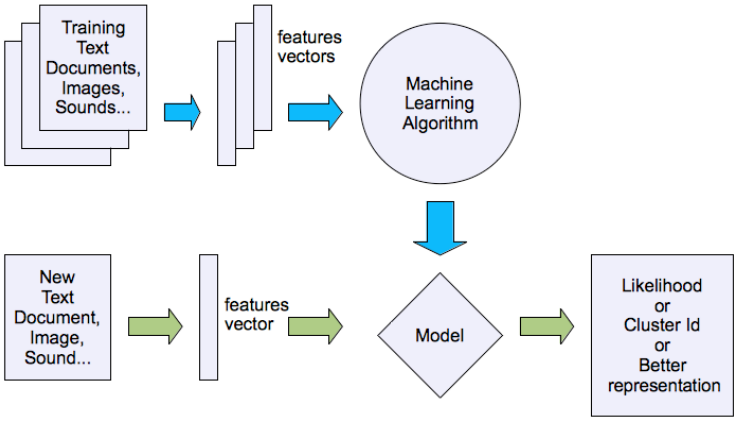
(ii) **Unsupervised learning**

Unsupervised learning, no labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end.

- **Clustering**

  In clustering, a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.

- **Probability distribution estimation**
- **Finding association (in features)**
- **Dimension reduction**
  - Dimensionality reduction simplifies inputs by mapping them into a lower-dimensional space. Topic modeling is a related problem, where a program is given a list of human language documents and is tasked to find out which documents cover similar topics.



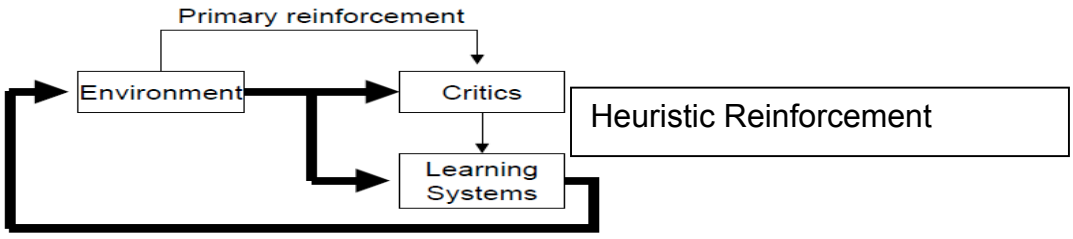
Example

- Suppose you have a basket and it is filled with some different types fruits, your task is to arrange them as groups.
- This time you don't know anything about the fruits, honestly saying this is the first time you have seen them. You have no clue about those.
- So, how will you arrange them? What will you do first???
- You will take a fruit and you will arrange them by considering physical character of that particular fruit.
- Suppose you have considered color.
    - Then you will arrange them on considering base condition as **color.**
    - Then the groups will be something like this.
        - RED COLOR GROUP: apples & cherry fruits.
        - GREEN COLOR GROUP: bananas & grapes.
- So now you will take another physical character such as **size**.
    - RED COLOR AND BIG SIZE: apple.
    - RED COLOR AND SMALL SIZE: cherry fruits.
    - GREEN COLOR AND BIG SIZE: bananas.
    - GREEN COLOR AND SMALL SIZE: grapes.
- Job done happy ending.
- Here you did not learn anything before, means no train data and no response variable.
- This type of learning is known as **unsupervised learning.**
- **Clustering** comes under **unsupervised learning**.

(iii) Reinforcement learning

In reinforcement learning, a computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle), without a teacher explicitly telling it whether it has come close to its goal or not. Another example is learning to play a game by playing against an opponent.

- Decision making (robot, chess machine)



7. Explain Adaptive learning with example.

Adaptive learning refers broadly to a learning process where the content taught or the way such content is presented changes or "adapts" based on the responses of the individual student.

Adaptive learning is an educational method which uses computers as interactive teaching devices, and to orchestrate the allocation of human and mediated resources according to the unique needs of each learner. Computers adapt the presentation of educational material according to students' learning needs, as indicated by their responses to questions, tasks and experiences. The technology encompasses aspects derived from various fields of study including computer science, education, psychology, and brain science.

Adaptive learning has been implemented in several kinds of educational systems such as adaptive educational hypermedia, intelligent tutoring systems, Computerized adaptive testing, and computer-based pedagogical agents, among others.

The **goal of an** adaptive learning system is to personalize instruction in order to improve or accelerate a student's performance again.

Adaptive learning systems have traditionally been divided into separate components or 'models'.

- Expert model - The model with the information which is to be taught
- Student model - The model which tracks and learns about the student
- Instructional model - The model which actually conveys the information

Expert model

The expert model stores information about the material which is being taught. This can be as simple as the solutions for the question set but it can also include lessons and tutorials and, in more sophisticated systems, even expert methodologies to illustrate approaches to the questions.

Adaptive learning systems which do not include an expert model will typically incorporate these functions in the instructional model.

Student model

Student model algorithms have been a rich research area over the past twenty years. The simplest means of determining a student's skill level is the method employed in CAT (Computerized adaptive testing). In CAT, the subject is presented with questions that are selected based on their level of difficulty in relation to the presumed skill level of the subject. As the test proceeds, the computer adjusts the subject's score based on their answers, continuously fine-tuning the score by selecting questions from a narrower range of difficulty.

An algorithm for a CAT-style assessment is simple to implement. A large pool of questions is amassed and rated according to difficulty, through expert analysis, experimentation, or a combination of the two. The computer then performs what is essentially a binary search, always giving the subject a question which is half way between what the computer has already determined to be the subject's maximum and minimum possible skill levels. These levels are then adjusted to the level of the difficulty of the question, reassigning the minimum if the subject answered correctly, and the maximum if the subject answered incorrectly. Obviously, a certain margin for error has to be built in to allow for scenarios where the subject's answer is not indicative of their true skill level but simply coincidental. Asking multiple questions from one level of difficulty greatly reduces the probability of a misleading answer, and allowing the range to grow beyond the assumed skill level can compensate for possible misevaluations.

Instructional model

The instructional model generally looks to incorporate the best educational tools that technology has to offer (such as multimedia presentations) with expert teacher advice for presentation methods. The level of sophistication of the instructional model depends greatly on the level of sophistication of the student model. In a CAT-style student model, the instructional model will simply rank lessons in correspondence with the ranks for the question pool. When the student's level has been satisfactorily determined, the instructional model provides the appropriate lesson. The more advanced student models which assess based on concepts need an instructional model which organizes its lessons by concept as well. The instructional model can be designed to analyze the collection of weaknesses and tailor a lesson plan accordingly.

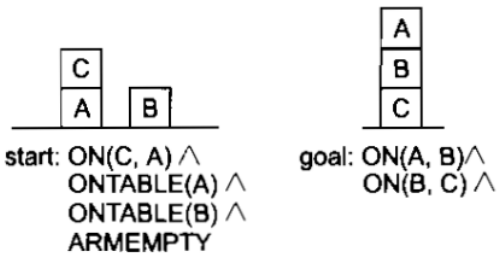
When the incorrect answers are being evaluated by the student model, some systems look to provide feedback to the actual questions in the form of 'hints'. As the student makes mistakes, useful suggestions pop up such as "look carefully at the sign of the number". This too can fall in the domain of the instructional model, with generic concept-based hints being offered based on concept weaknesses, or the hints can be question-specific in which case the student, instructional, and expert models all overlap.

8. Explain Non-linear planning.

The goal stack planning method attacks problems involving conjoined goals by solving the goals one at a time, in order. A plan generated by this method contains sequence of operators for attaining the first goal, followed by the complete sequence for the second goal. But, the difficult problems cause goal interactions.

The operators used to solve one sub problem may interfere with the solution to a previous sub problem. Most sub problems require an intertwined plan in which multiple sub-problems are worked on simultaneously. Such plan is called a **nonlinear plan** because it is not composed of a linear sequence of complete sub-plans.

Let us reconsider the **SUSSMAN ANOMALY**



A good plan for the solution of this problem is following

1. Try to achieve ON(A,B) clearing block A putting block C on the table.
2. Achieve ON(B,C) by stacking block B on block C.
3. Complete ON(A,B) by stacking block A on block B.

The initial plan consists of no steps and by studying the goal state ideas for the possible steps are generated. There is no order or detail at this stage. Gradually more detail is introduced and constraints about the order of subsets of the steps are introduced until a completely ordered sequence is created.

Tweak Heuristics Using Constraint Posting

The *Tweak* planning method involves the following heuristics.
Step Addition-- Creating new steps (GPS) for a plan.
Promotion-- Constraining a step to go before another step.
Declobbering-- Placing a new step between two steps to revert a precondition.
Simple Establishment-- Assigning a value to a variable to ensure a precondition.
Separation-- Preventing variables being assigned certain values.

In this problem means-end analysis suggests two steps with end conditions ON(A,B) and ON(B,C) which indicates the operator STACK giving the layout shown below where the operator is preceded by its preconditions and followed by its post conditions:

ON(A,B)	**ON(B,C)**
CLEAR(B)	CLEAR(C)
*HOLDING(A)	*HOLDING(B)
STACK(A,B)	STACK(B,C)
ARMEMPTY	ARMEMPTY
ON(A,B)	ON(B,C)
¬ CLEAR(B)	¬ CLEAR(C)
¬ HOLDING(A)	¬ HOLDING(B)

Many planning methods have introduced heuristics to achieve goals or preconditions. The TWEAK planning method brought all these together under one formalism. Other methods that introduced/used the following heuristics are mentioned in brackets in the following section.

CLEAR(A)	CLEAR(C)
ONTABLE(A)	ONTABLE(B)
*ARMEMPTY	*ARMEMPTY
PICKUP(A)	PICKUP(B)
¬ONTABLE(A)	¬ONTABLE(B)
¬ARMEMPTY	¬ARMEMPTY
HOLDING(A)	HOLDING(B)

In this case we need to state that a PICKUP step should precede a corresponding STACK step. That is to say

PICKUP(A) ⇐ STACK(A,B)
PICKUP(B) ⇐ STACK(B,C)

This gives four steps partially ordered and four unachieved conditions

- *CLEAR(A) -- block A is not clear in initial state.

- *CLEAR(B) -- although block B is clear in initial state STACK(A,B) with postcondition ¬CLEAR(B) might precede the step with *CLEAR(B) precondition.
- Two *ARMEMPTY -- initial state makes ARMEMPTY but PICKUP step has ¬ARMEMPTY and could again precede this step.

We can use the *promotion* heuristic to force one operator to precede another so that the postcondition of one operator STACK(A,B) does not negate the precondition CLEAR(B) of another operator PICKUP(B). This ordering is represented by

PICKUP(B) ⇽ STACK(A,B)

We can use promotion to achieve one of the ARMEMPTY preconditions:
Making PICKUP(B) precede PICKUP(A) ensures that the arm is empty and all the conditions for PICKUP(B) are met.
This is written

PICKUP(B) ⇽ PICKUP(A).

Unfortunately a postcondition of the first operator is that the arm becomes not empty, so we need to use the *declobbering* heuristic to achieve the preconditions of the second operator PICKUP(A).

Declobbering

- PICKUP(B) asserts ¬ARMEMPTY.
- But if we insert a step between PICKUP(B) and PICKUP(A) to reassert ARMEMPTY then we can achieve the precondition.
- STACK(B,C) can do this so we *post another constraint*:

PICKUP(B) ⇽ STACK(B,C) ⇽ PICKUP(A)

We still need to achieve CLEAR(A):
The appropriate operator is UNSTACK(x,A) by *step addition*. This leads to the following set of conditions

*CLEAR(x)
*ON(x,A)
*ARMEMPTY

UNSTACK(x,A)

¬ ON(x,A)
¬ ARMEMPTY
HOLDING(x)
CLEAR(A)

The variable x can be bound to the block C by the *simple establishment* heuristic since C is on A in the initial state. The preconditions CLEAR(C) and ARMEMPTY are negated by STACK(B,C) and by PICKUP(B) or PICKUP(A) however.

So we must introduce three orderings by *promotion* to ensure the operator UNSTACK(C,A).

UNSTACK(C,A) ⇽ STACK(B,C)
UNSTACK(C,A) ⇽ PICKUP(A)
UNSTACK(C,A) ⇽ PICKUP(B)

Promotion involves adding a step and this clobbers one of the preconditions of PICKUP(B) viz ARMEMPTY, always a potential problem with this heuristic.
However all is not lost as there is an operator, PUTDOWN that has the required postcondition and given that the operator UNSTACK(C,A) had generated the precondition for it of HOLDING(C) we can produce an extra operator successfully

HOLDING(C)

PUTDOWN(C)

¬ HOLDING(C)
ONTABLE(C)
ARMEMPTY

This operator *declobbers* the operator PICKUP(B) yielding the sequence

UNSTACK(C,A) ⇐ PUTDOWN(C) ⇐ PICKUP(B)

This yields the final sequence:

1. UNSTACK(C,A)
2. PUTDOWN(C)
3. PICKUP(B)
4. STACK (B,C)
5. PICKUP(A)
6. STACK(A,B)

Let us finish this section by looking at the formal form of the TWEAK algorithm:

Algorithm: Non Linear Planning (TWEAK)

1. Initialize S to be the set of propositions in the goal state.
2. Remove some unachieved proposition P from S.
3. Achieve P by using step addition, promotion, declobbering, simple establishment, or separation.
4. Review all the steps in the plan including any new steps introduced by step addition to see if any of their preconditions are unachieved. Add to S the new set of unachieved preconditions.
5. If s is empty, complete the plan by converting the partial order of steps in to a total order, and instantiate any variables as necessary.
6. Otherwise go to step 2.

9. Discuss hierarchical and reactive system in detail

HIERARCHICAL PLANNING

- To solve hard problems, problem solver may have to generate long plans.
- To do this efficiently, eliminate some of the details of the problem until a solution that addresses the main issues is found. Then an attempt can be made to fill in the appropriate details.

- ABSTRIPS is an approach, which actually planned in a hierarchy of abstraction spaces, in which each of the preconditions at a lower level of abstractions were ignored.

```
(OPERATOR
    (PRECONDITIONS
        (and(...)
            (forall (w ...)...)
            (not
                (exists ...)
                (or.....)))
    (POSTCONDITIONS
        (ADD (...))
        (DELETE (...))
        (if (and (...)(...))
            (ADD (...)(...))
            (DELETE (...)(...))))))
```

Example: Suppose we want to visit a friend in Europe, but you have a limited amount of cash to spend. It makes sense to check air fare first, since finding an affordable flight will be most difficult part of the task. You should not worry about getting your drive way, planning a route to airport, or parking your car until you are sure you have a flight.

The ABSTRIPS approach to problem solving is as follows:

- First solve the problem completely, considering only preconditions whose criticality value is the highest possible.
- Simply ignore preconditions of lower than peak criticality.
- Once this is done, use the constructed plan as the outline of a complete plan and consider preconditions at the next lowest criticality level.
- Augment the plan with operators that satisfy those preconditions.
- Again in choosing operators, ignore all preconditions whose criticality level is less than the level now being considered.
- Continue this process of considering less and less critical preconditions until all of the preconditions of the original rules have been considered.

Because this process explores entire plans at one level of detail before it looks at the lower level details of any of them. This is called **length-first search**.

REACTIVE SYSTEMS

Hierarchical planning and Non-linear planning systems are deliberative planning process, in which a plan for completing an entire task is constructed prior to action. The idea of reactive systems is to avoid planning altogether, and instead use the observable situation as a clue to which one can simply react.

It chooses actions one at a time; it does not anticipate and select an entire action sequence before it does the first thing.

- **Example**: Thermostat

The job of the thermostat is to keep temperature constant inside a room. The real thermostat uses simple pair of **situation-action rule.**

If the temperature in the room is k degrees above the desired temperature, then turn the air condition on.

If the temperature in the room is k degrees below the desired temperature, then turn the air condition off.

An intelligent system with limited resources must decide when to start thinking, when to stop thinking and when to act. Some mechanisms foe suspending plan execution is needed so that the system

can turn its attention to high priority goals. Finally some situations require immediate attention and rapid action. For this reason, some deliberative planners compile out relative sub systems based on their problem solving experiences.

Advantages:

1. Robustness – operate robustly in domains that are difficult to model completely and accurately
2. Extremely responsive – attractive for real time tasks like driving and walking.

10. Explain learning in problem solving.

Describes about problem solving can be performed without the aid of teacher.

Learning by Parameter Adjustment

Many programs rely on an evaluation procedure that combines information from several sources in to a single summary statistic. Pattern classification program often combine several features to determine the correct category in to which a given stimulus should be placed. In designing such programs, it is often difficult to know a priori how much weight should be attached to each feature being used. One way to finding the correct weights is to begin with some estimate of the correct settings and then to let program modify the settings on the basis of its experience. Features that appear to be good predictors of overall success will have their weights increased, while those that do not will have their weights decreased.

Learning with Macro-Operators

For example, suppose you are faced with the problem of getting to a downtown post office. Your solution may involve getting in your car, starting it and driving it along a certain route. Substantial planning may go in to choosing the appropriate route, but, you need not plan about how to go about starting your car. You are free to treat START-CAR as an atomic action, even though it really consists of several actions: sitting down, adjusting the mirror, inserting the key and turning the key. Sequence of actions that can be treated as a whole are called macro-operators. A MACROP is just like a regular operator except that it consists of sequence of actions, not just a single one.

Suppose we are given an initial blocks world situation in which ON(C, B) and ON(A, Table) are both true. STRIPS can achieve goal ON(A,B) by devising a plan with the four steps UNSTACK(C,B), PUTDOWN(C), PICKUP(A), STACK(A,B). STRIPS now build a MACROP with preconditions ON(C,B), ON(A, Table) and post conditions ON(C, Table), ON(A,B).

Learning by Chunking

Chunking is a process similar to macro-operators. Its computational basis is in production systems. SOAR also exploits chunking so that the performance can increase with experience. SOAR solve problem by firing productions which are stored in long term memory. When SOAR detects a useful sequence of production firings, it creates a chunk which is essentially a large production that does the work of an entire sequence of smaller ones.

SOAR learns how to place a given tile without permanently disturbing the previously placed tiles. Given the way that SOAR learns, several chunks may encode a single macro-operator, and one chunk may participate in a number of macro sequences. Chunks are generally applicable toward any goal state. Chunks learned during the initial stages of solving a problem are applicable in the later stages of the same problem solving episode. After a solution is found, the chunks remain in memory, ready for use in the next problem.

Unit- V
Part-A

1. **Define Expert systems.**

   An expert system is software that attempts to reproduce the performance of one or more human experts, most commonly in a specific problem domain. Developed via specialized software tools called shells.

   Ex: Diagnostic applications, servicing.

2. **Explain expert system development**
   - Problem definition
   - System design (knowledge acquisition)
   - Formalization (logical design, tree structures)
   - System implementation (building a prototype)
   - System validation

3. **What are the characteristics of Expert system?**
   - Inferential processes
     - Uses carious reasoning techniques
   - Heuristics
     - Decisions based on experience and knowledge
   - Waterman
     - Expertise
     - Depth
     - Symbolic reasoning
     - Self knowledge

4. **What are the limitations of Expert systems**
   - Not widely used or tested.
   - Limited to relatively narrow problems
   - Cannot readily deal with "mixed" knowledge
   - Possibility of error
   - Cannot refine own knowledge base
   - Difficult to maintain
   - May have high developed costs
   - Raise legal and ethical concerns

5. **What are the capabilities of Expert system.**
   - Explore impact of strategic goals
   - Impact of plans on resources
   - Integrated general design principles and manufacturing limitations\
   - Provide advice on decisions
   - Monitor quality and assist in finding solutions
   - Look for causes and suggest solutions

6. **What is a shell**

   A piece of software which contains:
   - The user interface
   - A format for declarative knowledge in the knowledge base
   - An interface engine

7. **What are the various types of Expert systems**
   - Rule base expert systems
   - Frame based expert systems
   - Hybrid systems

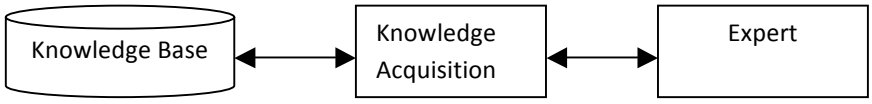
- Model based systems
- Ready-made systems
- Real-time expert systems

8. Mention the 3 major components of an Expert system

- Knowledge base
- Inference engine
- User interface

9. What is knowledge acquisition?

it provides convenient and efficient means of capturing and storing all components of the knowledge base.



10. What are the advantages of Expert systems

- Easy to develop and modify
- The use of satisficing
- The use of heuristics
- Development by knowledge engineers and users

11. Mention applications of Expert system.

- Credit granting
- Information management and retrieval
- Hospital and medical facilities
- Loan analysis
- Virus detection
- Repair and maintenance
- Warehouse optimization

12. What is heuristic?

Heuristics are formalized as rules for choosing those branches in a state space that are most likely to lead to an acceptance problem solution. Problem solver employ heuristics in two basic situations:

- A problem may not have an exact solution because of inherent ambiguity in the problem statement or available data.
- A problem may have an exact solution, but the computational cost of finding it may be prohibitive.

13. Define MYCIN

It is a program for treating blood infections. MYCIN is used to diagnose patients based on reported symptoms and medical test results. The program could further information concerning the patient, as well as suggest additional laboratory tests, to arrive at a probable diagnosis, after which it would recommend a course of treatment.

14. Define DART

DART (Dynamic Analysis and Replanning Tool) is an program used to optimize and schedule the transportation of supplies or personnel and solve other logistical problems.DART uses intelligent agents to aid decision support systems. It integrates a

set of intelligent data processing agents and database management systems to give planners the ability to rapidly evaluate plans for logistical feasibility.

15. Describe how Expert systems perform inference

The brain of an expert system is the inference engine that provides a methodology for reasoning about information in the knowledge base. Inference can be performed using semantics networks, production rules, and logic statements.

16. What are real time expert system

In real time expert system the conclusions are derived fast so a process can be impacted immediately. They are used in quality control and robotics(ex: to correct a mal function).

17. What are the classifications of expert systems?

Classification is based on expertness or purpose

- An assistant
- A colleague
- A true expert

18. Mention the features of Expert system.

- Dealing with uncertainty
  - Certainty factors
- Explanation
- Ease of modification
- Transportability
- Adaptive learning

19. Write the steps to create an Expert system

- Extracting knowledge and methods from the expert (knowledge acquisition).
- Reforming knowledge/methods into an organised form (knowledge representation).

20. What are the categories of knowledge

- Declarative
  - Descriptive, facts, shallow knowledge
- Procedural
  - Way things work, tell how to make inferences
- Semantic
  - Symbols
- Episodic
  - Autobiographical, experimental
- Meta-knowledge
  - Knowledge about the knowledge

Part- B

1. **What are Expert systems? Explain in detail.**

An expert is a person who has the expertise and knowledge of his specialized field. Heart specialist and a mathematics expert etc

Through experience, an expert expands his skills to enable him to solve problems heuristically, efficiently and effectively.

The process of ES development is iterative. Steps in developing the ES include −

Identify Problem Domain

- The problem must be suitable for an expert system to solve it.
- Find the experts in task domain for the ES project.
- Establish cost-effectiveness of the system.

Design the System

- Identify the ES Technology
- Know and establish the degree of integration with the other systems and databases.
- Realize how the concepts can represent the domain knowledge best.

Develop the Prototype

From Knowledge Base: The knowledge engineer works to −

- Acquire domain knowledge from the expert.
- Represent it in the form of If-THEN-ELSE rules.

Test and Refine the Prototype

- The knowledge engineer uses sample cases to test the prototype for any deficiencies in performance.
- End users test the prototypes of the ES.

Develop and Complete the ES

- Test and ensure the interaction of the ES with all elements of its environment, including end users, databases, and other information systems.
- Document the ES project well
- Train the user to use ES.

Examples of Expert System

Weather forecasting and radar projections use applications of expert systems. These forecasting systems use several rules based on temperature, wind, humidity, and jet stream. The application makes projections on potential weather patterns based on specific conditions and rules within the application.

The logistics management necessary to respond to natural disasters requires applications of expert systems. This software can help emergency personal determine where to place equipment for disaster relief resources. This is most often used in forest firefighting, hurricane relief, and earthquake relief around the world.

War-simulation programs are applications of expert systems. These programs use complex rules to determine armory logistics and human casualties during warfare situations. The war simulation programs are used to help governments determine the best approach to take during a confrontation.

Human intelligence and experience is what makes an expert system work. These problem-solving techniques are transformed into business rules that help the program weigh options on specific decisions. Each expert system requires human subject matter experts to convert logical process into specific rules.

The human genome project was a program that used expert systems to decipher human genetics. This provided advance mapping features to scientists that helped determine the

linkage between genetics and human characteristics. Without expert systems, this linkage would have been difficult to discover.

Three fundamental roles in building expert systems are:
1. *Expert* - Successful ES systems depend on the experience and application of knowledge that the people can bring to it during its development. Large systems generally require multiple experts.

2. *Knowledge engineer* - The knowledge engineer has a dual task. This person should be able to elicit knowledge from the expert, gradually gaining an understanding of an area of expertise. Intelligence, tact, empathy, and proficiency in specific techniques of knowledge acquisition are all required of a knowledge engineer. Knowledge-acquisition techniques include conducting interviews with varying degrees of structure, protocol analysis, observation of experts at work, and analysis of cases.
On the other hand, the knowledge engineer must also select a tool appropriate for the project and use it to represent the knowledge with the application of the ***knowledge acquisition facility***.

3. *User* - A system developed by an end user with a simple shell, is built rather quickly an inexpensively. Larger systems are built in an organized development effort. A prototype-oriented iterative development strategy is commonly used. ESs lends themselves particularly well to prototyping. In artificial intelligence, an **expert system** is a computer system that emulates the decision-making ability of a human expert. Expert systems are designed to solve complex problems by reasoning about knowledge, represented primarily as if–then rules rather than through conventional procedural code

2. Elaborately explain the process of knowledge acquisition.

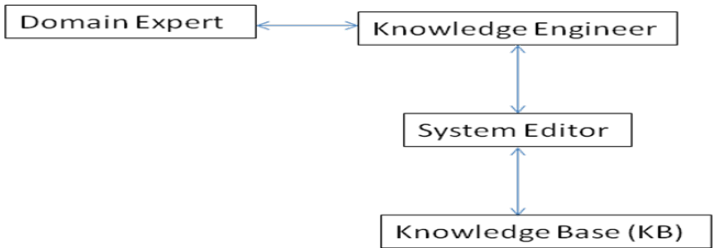
Knowledge acquisition is transferring knowledge from human expert to computer. Knowledge acquisition includes the elicitation, collection, analysis, modeling and validation of knowledge.

- Process is similar to rapid prototyping (expert is the customer)
- Expert is involved throughout the development process
- Incremental systems are presented to expert for feedback and approval
- Change is viewed as healthy not a process failure
- Expert may not have required knowledge in some areas
- Expert may not be consciously aware of required knowledge needed
- Expert may not be able to communicate the knowledge needed to knowledge engineer
- Knowledge engineer may not be able to structure knowledge for entry into knowledge base.

Steps in knowledge acquisition

1. Collect: (elicitation)
   - Getting the knowledge out of the expert
   - Most difficult step
   - Lots of strategies
2. Interpret:
   - Review collected knowledge, organize, and filter

3. Analyze:
   - Determining types of knowledge, conceptual relationships
   - Determining appropriate knowledge representations & inference structure
4. Design:
   - Extracting more knowledge after using above principles



Domain Expert

- Anyone can be considered a **domain expert** if he or she has deep knowledge (of both facts and rules) and strong practical experience in a particular domain. The area of the domain may be limited. In general, an expert is a skilful person who can do things other people cannot.Eg: Doctor
- Provides knowledge and processes needed to solve problem
- Domain expert must be available for hundreds of hours
- Knowledge in the expert system ends up being the knowledge engineer's understanding of the domain, not the domain expert's knowledge

Knowledge Engineer

Knowledge engineers are involved with validation and verification.

Validation is the process of ensuring that something is correct or conforms to a certain standard. A knowledge engineer is required to carry out data collection and data entry, but they must use validation in order to ensure that the data they collect, and then enter into their systems, fall within the accepted boundaries of the application collecting the data.

It is important that a knowledge engineer incorporates validation procedures into their systems within the program code. After the knowledge-based system is constructed, it can be maintained by the domain expert

System Editor

- Knowledge base editor which help the expert or knowledge engineer to easily update and check the knowledge base .

Knowledge Base:

- Knowledge about problem domain in the form of static and dynamic databases.
- Static knowledge consists of rules and facts which is complied as a part of the system and does not change during execution of the system.
- Dynamic knowledge consists of facts related to a particular consultation of the system.
  - At the beginning of the consultation, the dynamic knowledge base often called working memory is empty.
  - As a consultation progresses, dynamic knowledge base grows and is used along with static knowledge in decision making.
- Working memory is deleted at the end of consultation of the system.

Components of Knowledge Base

The knowledge base of an ES is a store of both, factual and heuristic knowledge.

- **Factual Knowledge** − It is the information widely accepted by the Knowledge Engineers and scholars in the task domain.
- **Heuristic Knowledge** − Expert systems apply heuristics to guide the reasoning and thus reduce the search area for a solution. It is about practice, accurate judgment, one's ability of evaluation, and guessing.

Truth Maintenance

- Task of maintaining the logical consistency of the rules in the rule-base
- Given the incremental manner in which rule-bases are built and since rules themselves are modular their interactions are hard to predict
- Newly added rules can render old rules obsolete and can be inconsistent with existing rules

1. Knowledge about components, e.g. voltage, ampere, priority, no of ports and so on.
2. Knowledge about constraints i.e. Rules for forming partial configuration of equipment's and extending successfully

Knowledge Acquisition Difficulties:

- Expert may not have required knowledge in some areas
- Expert may not be consciously aware of required knowledge needed
- Expert may not be able to communicate the knowledge needed to knowledge engineer Knowledge engineer may not be able to structure knowledge for entry into knowledge base

3. i) Explain the various stages of Expert system development.

Steps to Develop an Expert Systems

The process of ES development is iterative. Steps in developing the ES include −

Identify Problem Domain

- The problem must be suitable for an expert system to solve it.
- Find the experts in task domain for the ES project.
- Establish cost-effectiveness of the system.

Design the System

- Identify the ES Technology
- Know and establish the degree of integration with the other systems and databases.
- Realize how the concepts can represent the domain knowledge best.

Develop the Prototype

From Knowledge Base: The knowledge engineer works to −

- Acquire domain knowledge from the expert.
- Represent it in the form of If-THEN-ELSE rules.

Test and Refine the Prototype

- The knowledge engineer uses sample cases to test the prototype for any deficiencies in performance.
- End users test the prototypes of the ES.

Develop and Complete the ES

- Test and ensure the interaction of the ES with all elements of its environment, including end users, databases, and other information systems.
- Document the ES project well
- Train the user to use ES.

Examples of Expert System

Weather forecasting and radar projections use applications of expert systems. These forecasting systems use several rules based on temperature, wind, humidity, and jet stream. The application makes projections on potential weather patterns based on specific conditions and rules within the application.

The logistics management necessary to respond to natural disasters requires applications of expert systems. This software can help emergency personal determine where to place equipment for disaster relief resources. This is most often used in forest firefighting, hurricane relief, and earthquake relief around the world.

War-simulation programs are applications of expert systems. These programs use complex rules to determine armory logistics and human casualties during warfare situations. The war simulation programs are used to help governments determine the best approach to take during a confrontation.

Human intelligence and experience is what makes an expert system work. These problem-solving techniques are transformed into business rules that help the program weigh options on specific decisions. Each expert system requires human subject matter experts to convert logical process into specific rules.

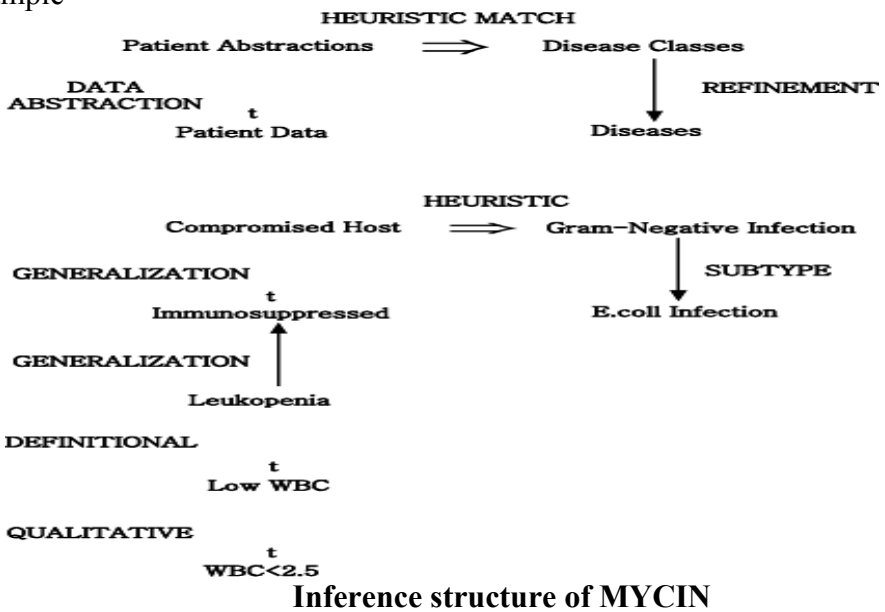
The human genome project was a program that used expert systems to decipher human genetics. This provided advance mapping features to scientists that helped determine the linkage between genetics and human characteristics. Without expert systems, this linkage would have been difficult to discover.

ii) Explain heuristics with a example

Expert systems apply heuristics to guide the reasoning and thus reduce the search area for a solution. It is about practice, accurate judgment, one's ability of evaluation, and guessing.

Heuristic classification

In simple classification, data may directly match solution features or may match after being abstracted. In heuristic classification, solutions and solution features may also be matched heuristically, by direct, non-hierarchical association with some concept in another classification hierarchy. For example, MYCIN does more than identify an unknown organism in terms of visible features of an organism: MYCIN heuristically relates an abstract characterization of the patient to a classification of diseases. We show this inference structure schematically, followed by an example

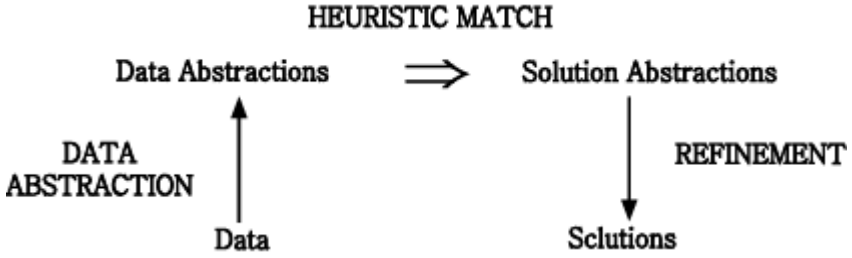


Inference structure of MYCIN

Basic observations about the patient are abstracted to patient categories, which are heuristically linked to diseases and disease categories. While only a subtype link with E.coli infection is shown here, evidence may actually derive from a combination of inferences. Some data might directly match E.coli features (an individual organism shaped like a rod and producing a Gram-negative stain is seen growing in a culture taken from the patient). Descriptions of laboratory cultures (describing location, method of collection, and incubation) can also be related to the classification of diseases.

The important link we have added is a heuristic association between a characterization of the patient (¡°compromised host¡±) and categories of diseases (¡°gram-negative infection¡±). Unlike definitional and hierarchical inferences, this inference makes a great leap. A heuristic relation is uncertain, based on assumptions of typicality, and is sometimes just a poorly understood correlation. A heuristic is often empirical, deriving from problem-solving experience; heuristics correspond to the ¡°rules of thumb,¡± often associated with expert systems (Feigenbaum, 1977).

Heuristics of this type reduce search by skipping over intermediate relations. These associations are usually uncertain because the intermediate relations may not hold in the specific case. Intermediate relations may be omitted because they are unobservable or poorly understood. In a medical diagnosis program, heuristics typically skip over the causal relations between symptoms and diseases. To summarize, in heuristic classification abstracted data statements are associated with specific problem solutions or features that characterize a solution. This can be shown schematically in simple terms.



Inference structure of heuristic classification

This diagram summarizes how a distinguished set of terms (data, data abstractions, solution abstractions, and solutions) are related systematically by different kinds of relations. This is the structure of inference in heuristic classification.

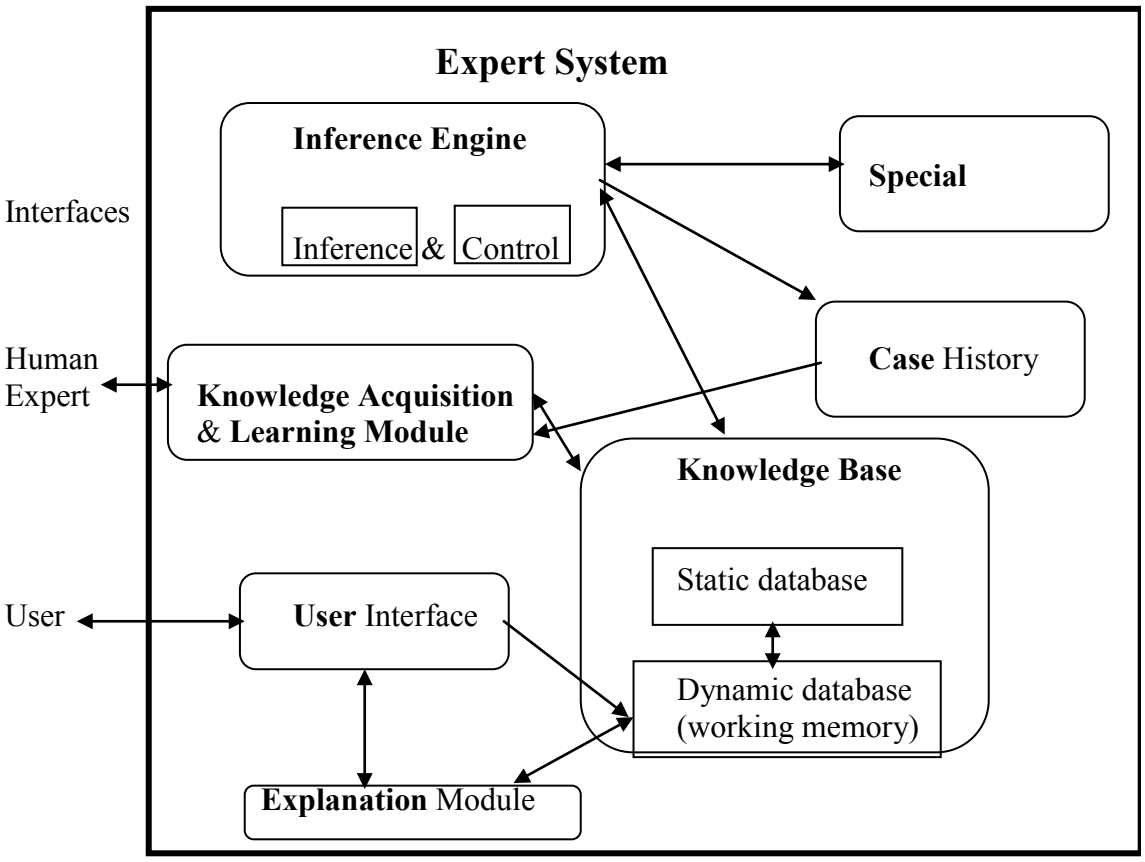
Criteria of Selecting Problem

- Recognized experts exist
- Experts do better than amateur
- Expert needs significant time to solve it
- Cognitive type tasks
- Skill can routinely taught to beginners
- Domain has high payoff
- Task does not require common sense

4. **Draw the schematic diagram of an expert system. Explain all the relevant components**.

ARCHITECTURE OF EXPERT SYSTEM

..



Knowledge Base:

- Knowledge about problem domain in the form of static and dynamic databases.
- Static knowledge consists of rules and facts which is complied as a part of the system and does not change during execution of the system.
- Dynamic knowledge consists of facts related to a particular consultation of the system.
  - At the beginning of the consultation, the dynamic knowledge base often called working memory is empty.
  - As a consultation progresses, dynamic knowledge base grows and is used along with static knowledge in decision making.
- Working memory is deleted at the end of consultation of the system.

Components of Knowledge Base

The knowledge base of an ES is a store of both, factual and heuristic knowledge.

- **Factual Knowledge** − It is the information widely accepted by the Knowledge Engineers and scholars in the task domain.
- **Heuristic Knowledge** − It is about practice, accurate judgement, one's ability of evaluation, and guessing.

Inference Engine:

- It consists of inference mechanism and control strategy.
- Inference means search through knowledge base and derive new knowledge.

- It involve formal reasoning involving matching and unification similar to the one performed by human expert to solve problems in a specific area of knowledge.
- Inference operates by using modus ponen rule.
- Control strategy determines the order in which rules are applied.
- There are mainly two types of control mechanism viz., forward chaining and backward chaining.

To recommend a solution, the interface engine uses the following strategies −

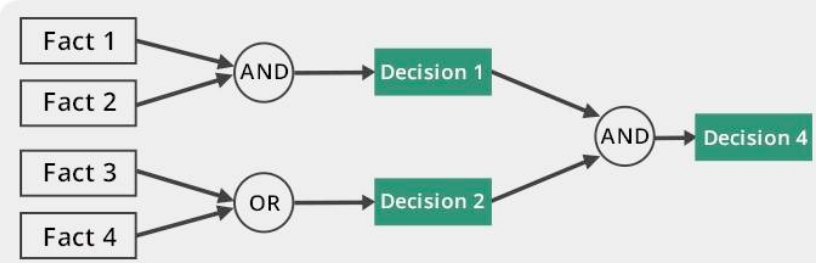
- Forward Chaining
- Backward Chaining

Forward Chaining

It is a strategy of an expert system to answer the question, **"What can happen next?"**

Here, the interface engine follows the chain of conditions and derivations and finally deduces the outcome. It considers all the facts and rules, and sorts them before concluding to a solution.

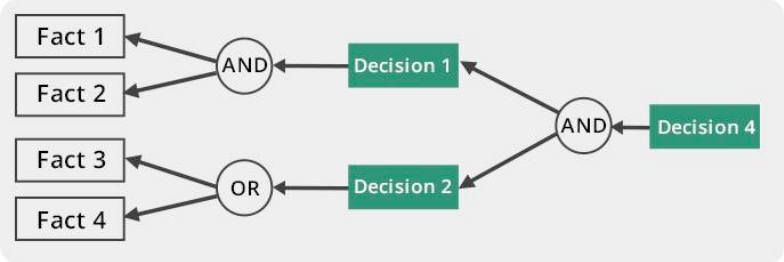
This strategy is followed for working on conclusion, result, or effect. For example, prediction of share market status as an effect of changes in interest rates.



Backward Chaining

With this strategy, an expert system finds out the answer to the question, **"Why this happened?"**

On the basis of what has already happened, the interface engine tries to find out which conditions could have happened in the past for this result. This strategy is followed for finding out cause or reason. For example, diagnosis of blood cancer in humans.



Knowledge Acquisition:

- Knowledge acquisition module allows system to acquire knowledge about the problem domain.
- Sources of Knowledge for ES
  - Text books, reports, case studies,
  - Empirical data and
  - Domain expert experience.
- Updating of Knowledge can be done using knowledge acquisition module of the system.
  - Insertion,

- Deletion and
- Updation of existing knowledge

Case History

- Case History stores the file created by inference engine using the dynamic database created at the time of consultation.
- Useful for learning module to enrich its knowledge base.
- Different cases with solutions are stored in Case Base system.
- These cases are used for solving problem using Case Base Reasoning (CBR).

Explanation Module

- Most expert systems have explanation facilities that allow the user to ask the system ***why*** it asked some question, and ***how*** it reached to conclusion.
- It contains 'How' and 'Why' modules attached to it.
  - The sub-module 'How' tells the user about the process through which system has reached to a particular solution
  - 'Why' sub-module tells that why is that particular solution offered.
- It explains user about the reasoning behind any particular problem solution.
- Questions are answered by referring to the system goals, the rules being used, and any existing problem data.

User Interfaces:

Allows user to communicate with system in interactive mode and helps system to create working knowledge for the problem to be solved. The explanation may appear in the following forms −

- Natural language displayed on screen.
- Verbal narrations in natural language.
- Listing of rule numbers displayed on the screen.

Dialogue Module (User Interface)	
System	Do you have fever?
User	Yes
System	Do you have bad throat?
User	No
System	Do you have cough?
User	Yes
System	Are you suffering from running nose?
User	Yes
System	Are you suffering from headache?
User	No

Special Interfaces:

- It may be used for specialized activities such as handling uncertainty in knowledge.
- This is a major area of expert systems research that involves methods for reasoning with uncertain data and uncertain knowledge.
- Knowledge is generally incomplete and uncertain.
- To deal with uncertain knowledge, a rule may have associated with it a *confidence factor* or a weight.
- The set of methods for using uncertain knowledge in combination with uncertain data in the reasoning process is called reasoning with uncertainty.

5. i) Explain the components of expert systems with a neat diagram.

Three fundamental roles in building expert systems are:

1. *Expert* - Successful ES systems depend on the experience and application of knowledge that the people can bring to it during its development. Large systems generally require multiple experts.

2. *Knowledge engineer* - The knowledge engineer has a dual task. This person should be able to elicit knowledge from the expert, gradually gaining an understanding of an area of expertise. Intelligence, tact, empathy, and proficiency in specific techniques of knowledge acquisition are all required of a knowledge engineer. Knowledge-acquisition techniques include conducting interviews with varying degrees of structure, protocol analysis, observation of experts at work, and analysis of cases.

On the other hand, the knowledge engineer must also select a tool appropriate for the project and use it to represent the knowledge with the application of the ***knowledge acquisition facility***.

3. *User* - A system developed by an end user with a simple shell, is built rather quickly an inexpensively. Larger systems are built in an organized development effort. A prototype-oriented iterative development strategy is commonly used. ESs lends themselves particularly well to prototyping. In artificial intelligence, an **expert system** is a computer system that emulates the decision-making ability of a human expert. Expert systems are designed to solve complex problems by reasoning about knowledge, represented primarily as if–then rules rather than through conventional procedural code

Domain Expert

- Anyone can be considered a **domain expert** if he or she has deep knowledge (of both facts and rules) and strong practical experience in a particular domain. The area of the domain may be limited. In general, an expert is a skilful person who can do things other people cannot.Eg: Doctor
- Provides knowledge and processes needed to solve problem
- Domain expert must be available for hundreds of hours
- Knowledge in the expert system ends up being the knowledge engineer's understanding of the domain, not the domain expert's knowledge

System Editor

Knowledge base editor which help the expert or knowledge engineer to easily update and check the knowledge base .

Knowledge Base:

Knowledge about problem domain in the form of static and dynamic databases.

Static knowledge consists of rules and facts which is complied as a part of the system and does not change during execution of the system.

Dynamic knowledge consists of facts related to a particular consultation of the system.

At the beginning of the consultation, the dynamic knowledge base often called working memory is empty.

- As a consultation progresses, dynamic knowledge base grows and is used along with static knowledge in decision making.
- Working memory is deleted at the end of consultation of the system.

Components of Knowledge Base

The knowledge base of an ES is a store of both, factual and heuristic knowledge.

Factual Knowledge − It is the information widely accepted by the Knowledge Engineers and scholars in the task domain.

Heuristic Knowledge − Expert systems apply heuristics to guide the reasoning and thus reduce the search area for a solution. It is about practice, accurate judgment, one's ability of evaluation, and guessing.

Truth Maintenance

- Task of maintaining the logical consistency of the rules in the rule-base
- Given the incremental manner in which rule-bases are built and since rules themselves are modular their interactions are hard to predict
- Newly added rules can render old rules obsolete and can be inconsistent with existing rules
    - Knowledge about components, e.g. voltage, ampere, priority, no of ports and so on.
    - Knowledge about constraints i.e. Rules for forming partial configuration of equipment's and extending successfully

ii) Discuss the features of Expert systems

- An Expert System is built because of two factors: either to replace or to help an expert.
- Reducing operational costs. To hire an expert is costly.
- To replace a retiring or a leaving employee who is an expert.
- Help experts in their routine to improve productivity.
- Help experts in their more complex and difficult tasks
- An expert to obtain information needed by other experts who have forgotten about it or who are too busy to search for it.
- Characteristics of knowledge and inference engines used by domain experts
- Time and money spent on the project.
- Programming Capabilities available in-house.
- Hardware available for development
- Hardware available for deployment
- Required Performance of the system.

The main areas of application of ES are (Waterman, 1986)

Interpretation — drawing high–level conclusions based on data.
Prediction — projecting probable outcomes.
Diagnosis — determining the cause of malfunctions, disease, etc.
Design — finding best configuration based on criteria.
Planning — proposing a series of actions to achieve a goal.
Monitoring — comparing observed behaviour to the expected behaviour.
Debugging and Repair — prescribing and implementing remedies.
Instruction — assisting students in learning.
Control — governing the behaviour of a system

6. Discuss the advantages and limitations of expert systems.

What is Expert System?

- An expert is a person who has the expertise and knowledge of his specialized field. Heart specialist and a mathematics expert etc
- Through experience, an expert expands his skills to enable him to solve problems heuristically, efficiently and effectively.

What is Expertise?

- Expertise is the extensive, task-specific knowledge acquired from training, reading and experience
- Enables experts to be better and faster than non experts

Advantages

- Increased productivity (find solutions much faster than humans).
- Availability of expertise (human experts can be at one place at a time).
- Can be used in dangerous environments (e.g. in space).
- Reuse of User Interface and Inference modules
- Level of programming skill needed is low.
- Faster and cheaper completion of project

ES covers the following list of problem types in ES applications.

Interpretation: An interpretation system takes observed data and explains its meaning by inferring the problem state which corresponds to the observed data. Examples are Dipmeter Advisor, a system for interpreting geophysical oil well log data, and Prospector, a system for identifying geological ore-bearing formations.

Diagnosis: Diagnosis systems infer malfunctions or system state from observed irregularities and interpretation of data. MYCIN, an infectious disease diagnostician and several other medical diagnosis programs fall into this category.

Monitoring: A monitor observes system behavior and compares the observations to the planned behavior to determine flaws in the plan or potential malfunctions of the system. An example is Ventilation Manager, a program for monitoring a patient's ventilation therapy.

Design: Design is the process of developing a configuration for an object which satisfies all applicable constraints. R1 (or XCON) is an example of a design system which is used to configure VAX4 computers.

Planning: Planning is a design process that yields a set a actions intended to produce a desired outcome. An example is MOLGEN, a KBES for planning experiments in molecular genetics.

Limitations

- Difficulty in engineering, especially acquiring the expertise.
- Mistrust by the users.
- Effective only in specific areas (areas of expertise)

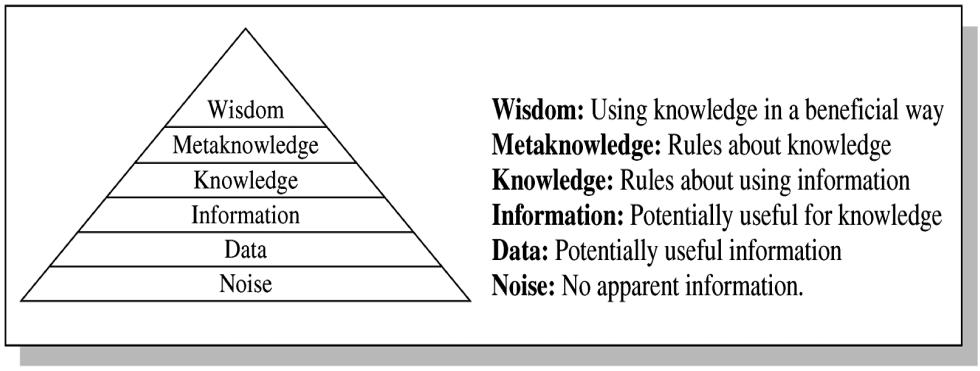
7. i) Explain briefly about Meta knowledge

Basic Concepts

- Knowledge is power, often inexact & incomplete and often poorly specified
- Amateurs become experts slowly
- Expert systems must be flexible and transparent
- Separate inference engine and knowledge base (make system easy to modify)
- Use uniform "fact" representation (reduces number of rules required and limits combinatorial explosion)
- Keep inference engine simple (makes knowledge acquisition and truth maintenance easier)
- Exploit redundancy (can help overcome problems due to inexact or uncertain reasoning)

Meta Knowledge

- Meta knowledge is knowledge about knowledge and expertise.
- Most successful expert systems are restricted to as small a domain as possible.
- In an expert system, ontology is the Meta knowledge that describes everything known about the problem domain.
- Wisdom is the Meta knowledge of determining the best goals of life and how to obtain them.



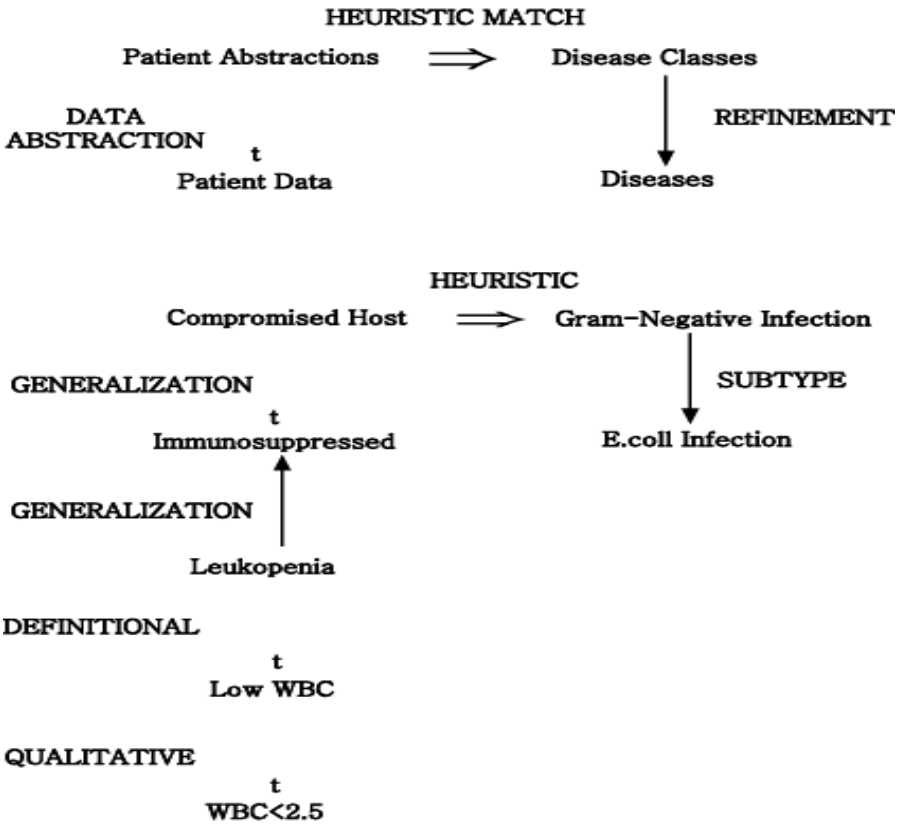
Pyramid of Knowledge

Heuristics

Expert systems apply heuristics to guide the reasoning and thus reduce the search area for a solution. It is about practice, accurate judgment, one's ability of evaluation, and guessing.

Heuristic classification

In simple classification, data may directly match solution features or may match after being abstracted. In heuristic classification, solutions and solution features may also be matched heuristically, by direct, non-hierarchical association with some concept in another classification hierarchy. For example, MYCIN does more than identify an unknown organism in terms of visible features of an organism: MYCIN heuristically relates an abstract characterization of the patient to a classification of diseases. We show this inference structure schematically, followed by an example



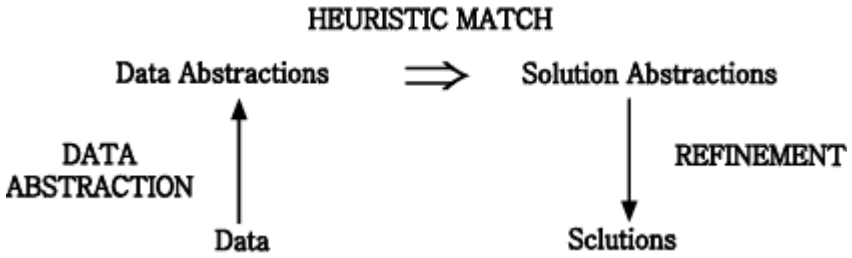
Inference structure of MYCIN

Basic observations about the patient are abstracted to patient categories, which are heuristically linked to diseases and disease categories. While only a subtype link with E.coli infection is shown here, evidence may actually derive from a combination of inferences. Some data might directly match E.coli features (an individual organism shaped like a rod and producing a Gram-negative stain is seen growing in a culture taken from the patient). Descriptions of laboratory cultures (describing location, method of collection, and incubation) can also be related to the classification of diseases.

The important link we have added is a heuristic association between a characterization of the patient (¡°compromised host¡±) and categories of diseases (¡°gram-negative infection¡±). Unlike definitional and hierarchical inferences, this inference makes a great leap. A heuristic relation is uncertain, based on assumptions of typicality, and is sometimes just a poorly understood correlation. A heuristic is often empirical, deriving from problem-solving experience; heuristics correspond to the ¡°rules of thumb,¡± often associated with expert systems (Feigenbaum, 1977).

Heuristics of this type reduce search by skipping over intermediate relations. These associations are usually uncertain because the intermediate relations may not hold in the specific case. Intermediate relations may be omitted because they are unobservable or poorly understood. In a medical diagnosis program, heuristics typically skip over the causal relations between symptoms and diseases. To summarize, in heuristic classification

abstracted data statements are associated with specific problem solutions or features that characterize a solution. This can be shown schematically in simple terms.



Inference structure of heuristic classification

This diagram summarizes how a distinguished set of terms (data, data abstractions, solution abstractions, and solutions) are related systematically by different kinds of relations. This is the structure of inference in heuristic classification.

Criteria of Selecting Problem

- Recognized experts exist
- Experts do better than amateur
- Expert needs significant time to solve it
- Cognitive type tasks
- Skill can routinely taught to beginners
- Domain has high payoff
- Task does not require common sense

ii) Explain the role of expert system

ROLES OF EXPERT SYSTEM

Three fundamental roles in building expert systems are:

1. *Expert* - Successful ES systems depend on the experience and application of knowledge that the people can bring to it during its development. Large systems generally require multiple experts.

2. *Knowledge engineer* - The knowledge engineer has a dual task. This person should be able to elicit knowledge from the expert, gradually gaining an understanding of an area of expertise. Intelligence, tact, empathy, and proficiency in specific techniques of knowledge acquisition are all required of a knowledge engineer. Knowledge-acquisition techniques include conducting interviews with varying degrees of structure, protocol analysis, observation of experts at work, and analysis of cases.

On the other hand, the knowledge engineer must also select a tool appropriate for the project and use it to represent the knowledge with the application of the ***knowledge acquisition facility***.

3. *User* - A system developed by an end user with a simple shell, is built rather quickly an inexpensively. Larger systems are built in an organized development effort. A prototype-oriented iterative development strategy is commonly used. ESs lends themselves

particularly well to prototyping. In artificial intelligence, an **expert system** is a computer system that emulates the decision-making ability of a human expert. Expert systems are designed to solve complex problems by reasoning about knowledge, represented primarily as if–then rules rather than through conventional procedural code

8. Write short notes on:

a. MYCIN and its applications

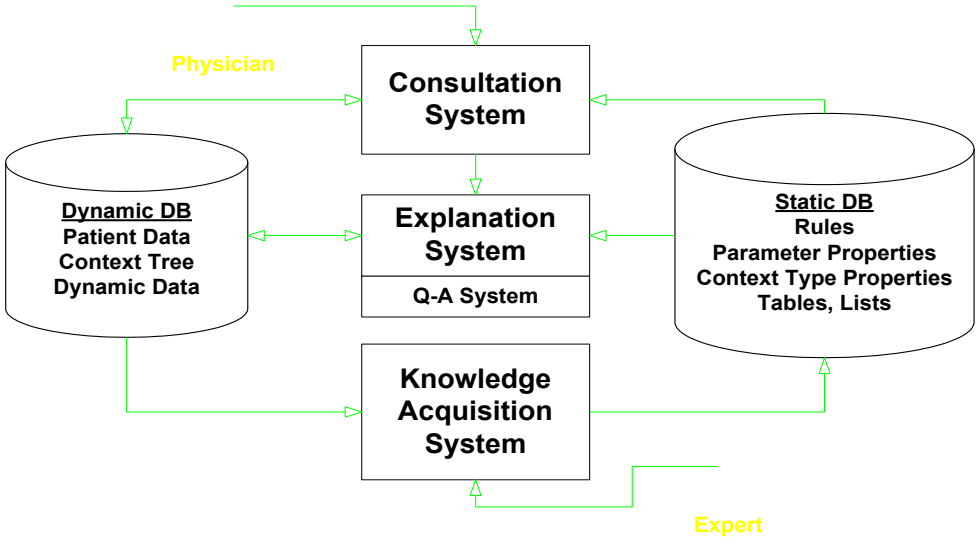
b. DART and its applications

i) MYCIN

Mycin operated using a fairly simple inference engine, a knowledge base of ~500 rules. It would query the physician running the program via a long series of simple yes/no or textual questions. At the end, it provided a list of possible culprit bacteria ranked from high to low based on the probability of each diagnosis, its confidence in each diagnosis' probability, the reasoning behind each diagnosis (that is, Mycin would also list the questions and rules which led it to rank a diagnosis a particular way), and its recommended course of drug treatment.

Despite Mycin's success, it is sometimes now used as a cautionary tale in AI courses for people who generate their own ad hoc probability frameworks. Mycin had a limited depth to its reasoning hierarchy due to the noise introduced by its certainty factor system. This problem can be solved by using a rigorous probabilistic framework, such as Bayesian statistics.

Architecture:



- Performs Diagnosis and Therapy Selection
- Control Structure reads Static DB (rules) and read/writes to Dynamic DB (patient, context)
- Linked to Explanations
- Terminal interface to Physician

Static Database

- Rules
- Meta-Rules
- Templates
- Rule Properties

- Context Properties
- Fed from Knowledge Acquisition System

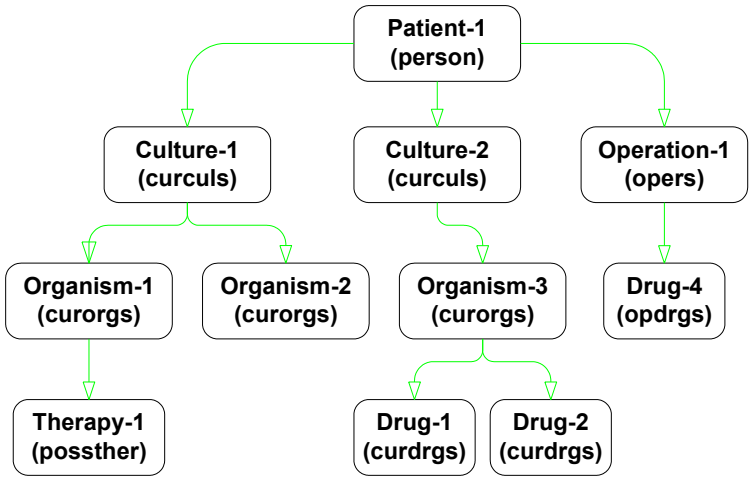
Dynamic Database

- Rules
- Meta-Rules
- Templates
- Rule Properties
- Context Properties
- Fed from Knowledge Acquisition System

Explanation System

- Provides reasoning why a conclusion has been made, or why a question is being asked
- Q-A Module
- Reasoning Status Checker
- Extends Static DB via Dialogue with Experts
- Dialogue Driven by System
- Requires minimal training for Experts

Context Tree



ii) DART

- In anticipation of computer faults, most manufacturers prepare specialized diagnostic aids that allow field engineers without complete knowledge of a system to diagnose the majority of its failures. Unfortunately, these diagnostics are not infallible for they do not take into account every fault and combination of faults for every possible system configuration. Thus, in some cases, it is necessary to call in someone with expert knowledge about the design of the system. These experts are expensive and often not immediately available, and there is inevitably a delay and a loss of working time to the system users.
- DART is a joint project of the Heuristic Programming Project and IBM that explores the application of artificial intelligence techniques to the diagnosis of computer faults. The

primary goal of the DART Project is to develop programs that capture the special design knowledge and diagnostic abilities of these experts and to make them available to field engineers. The practical goal is the construction of an automated diagnostician capable of pinpointing the functional units responsible for observed malfunctions in arbitrary system configurations.

9. Write notes on:

i) Expert system shell

Expert system shells are the software containing an interface, an inference engine, and the formatted skeleton of a knowledge base. In essence, an **expert system shell** is an empty bowl to be filled with the **expert** knowledge elements that the inference engine may process for users.

The E.S shell simplifies the process of creating a knowledge base. It is the shell that actually processes the information entered by a user relates it to the concepts contained in the knowledge base and provides an assessment or solution for a particular problem. Thus E.S shell provides a layer between the user interface and the computer O.S to manage the input and output of the data. It also manipulates the information provided by the user in conjunction with the knowledge base to arrive at a particular conclusion.

Knowledge Representation

Expert systems shells provide the bare bones for imitation of human expert reasoning in rule methods known as forward chaining and backward chaining.

Forward chaining in these shells enables taking data from a user and using inference engine rules to locate more data relative to that information until there is enough information to form a conclusion. Because the initial data received is what drives the seeking, this method is called a data driven method. An application that illustrates this forward-chaining method might explore the possibilities of arrangement of components within a computer to arrive at the best placement of the components.

Backward chaining gathers data only as it needs it when a knowledge base is being queried on a consultation. It has the goal of finding a value for C and reasons backward to discover the value of A and B that conclude the goal value of C. This method of reasoning from present data to prior data that was the underpinning of present data is called goal-driven method. An application illustrating expert system shells rules of inference might include a medical doctor inputting a current set of symptoms for background information on the same or similar symptoms in background information from a particular medical diagnosis expert system.

Inferred knowledge is gained by examination of existing facts to arrive at likely new information. This is the reasoning process that inhabits the inference engine in expert system shells.

ii) Limitations of Expert systems

- Difficulty in engineering, especially acquiring the expertise.
- Mistrust by the users.
- Effective only in specific areas (areas of expertise)

10. Explain the types of expert system architecture.

ES Architecture broadly classified as two types. They are as follows:

1. Production System Architectures (Or) Rule Based System Architectures

2. Non- Production System Architectures

i) Associative/Semantic Network Architectures
ii) Frame Architectures
iii) Decision Tree Architectures
iv) Blackboard System Architectures
v) Analogical Reasoning Architecture
vi) Neural Network Architectures

1. Production System Architecture (Or) Rule Based System

Production systems (or rule–based systems) are programs that instead of conventional algorithms use sets of IF–THEN rules (production rules). Unlike in algorithms, the order in which these rules should be used is not specified. It is decided by the program itself with respect to a problem state. In 1943, Post proved that any computable problem can be implemented in a production system. Cognitive scientists became interested in production systems because they seemed to represent better the way humans think and solve problems.

- Also known as "production systems" or "expert systems"
- Rule-based systems are one of the most successful AI paradigms
- Used for synthesis (construction) type systems
- Also used for analysis (diagnostic or classification) type systems

Label R_n if *condition1*
condition2
...
then
action1
action2 ...

Examples:

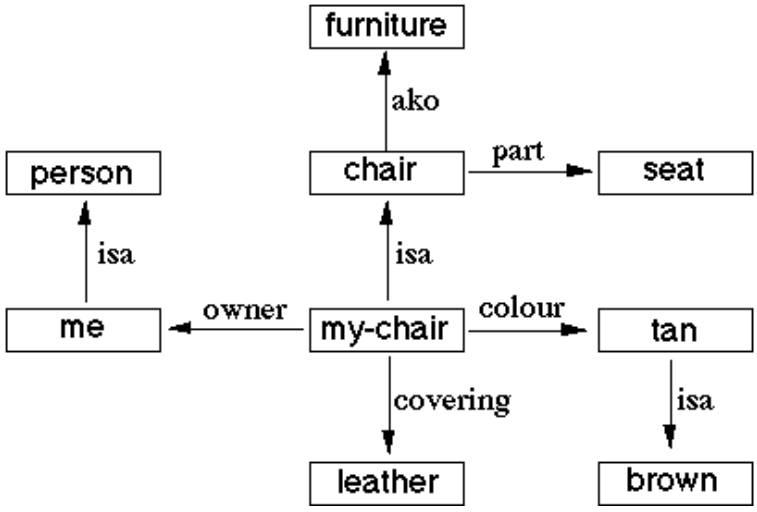
IF Saturday OR Sunday THEN go to cinema
IF NOT (Saturday OR Sunday) THEN go to work IF go to cinema THEN go outside
IF go to work AND NOT at work THEN go outside
IF NOT (can go outside) THEN stay home
IF good weather THEN can go outside
IF raining THEN have an umbrella
IF raining AND have an umbrella THEN can go outside

2. Non- Production System Architectures

i) Frame Architecture

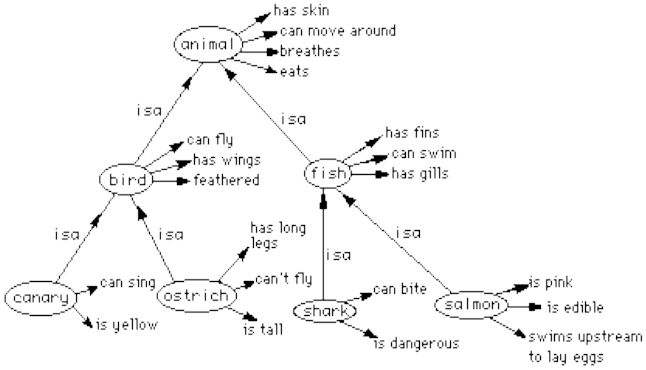
- One type of schema is a frame (or script – time-ordered sequence of frames).
- Frames are useful for simulating commonsense knowledge.
- Semantic nets provide 2-dimensional knowledge; frames provide 3-dimensional.
- Frames represent related knowledge about narrow subjects having much default knowledge
- The attributes, values and procedures are stored in specified slots.

- Slots size may vary.
- Example: PIP (Present Illness Program)
- Medical Expert System.
- Patient findings matched against frames, a trigger status occur.
- A frame is a group of slots and fillers that defines a stereotypical object that is used to represent generic / specific knowledge.
- Commonsense knowledge is knowledge that is generally known.
- Prototypes are objects possessing all typical characteristics of whatever is being modeled.
- Problems with frames include allowing unrestrained alteration / cancellation of slots.



ii) Associative/ Semantic Network Architectures

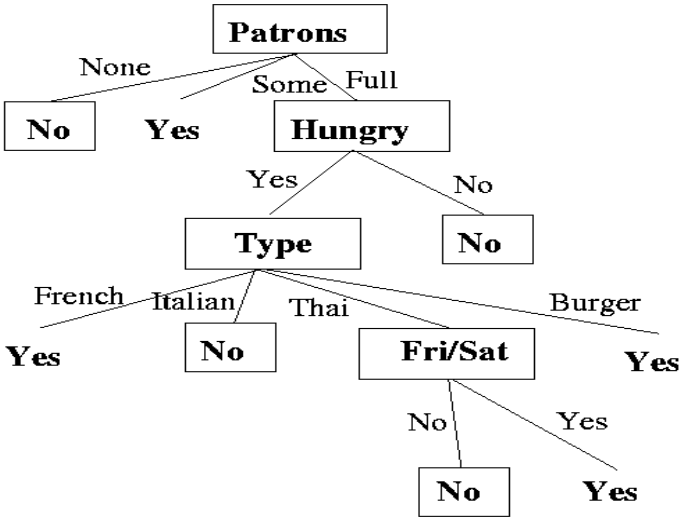
- A classic representation technique for propositional information
- Propositions – a form of declarative knowledge, stating facts (true/false)
- Propositions are called "atoms" – cannot be further subdivided.
- Semantic nets consist of nodes (objects, concepts, situations) and arcs (relationships between them**).**
- **Common type of links are..**
  IS-A – relates an instance or individual to a generic class
  A-KIND-OF – relates generic nodes to generic nodes



iii) Decision Tree Architecture

- Knowledge for ES may be stored in the form of Decision tree.
- Special tree building editor used.

- New nodes will be added when additional rules are added.
- Traversing technique to identify the attributes.

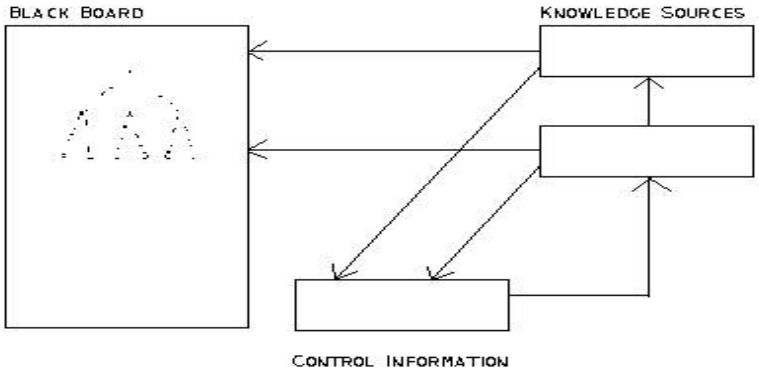


iv) Black Board Architecture

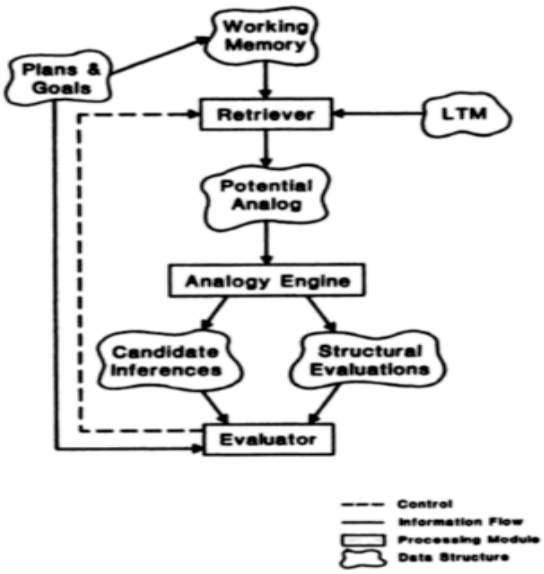
- There are 3 components
- There are number of *knowledge sources*.
- A data structure called *Blackboard* which contains current problem state and information needed by the knowledge sources.
- *Control Information* – Monitors changes in the blackboard.

A blackboard-system application consists of three major components

1. The software specialist modules, which are called **knowledge sources (KSs)**. Like the human experts at a blackboard, each knowledge source provides specific expertise needed by the application.
2. The **blackboard**, a shared repository of problems, partial solutions, suggestions, and contributed information. The blackboard can be thought of as a dynamic "library" of contributions to the current problem that have been recently "published" by other knowledge sources.
3. The **control shell**, which controls the flow of problem-solving activity in the system. Just as the eager human specialists need a moderator to prevent them from trampling each other in a mad dash to grab the chalk, KSs need a mechanism to organize their use in the most effective and coherent fashion. In a blackboard system, this is provided by the control shell.



v) Analogical Reasoning Architecture
Architecture:



Analogy-based reasoning: This term is sometimes used, as a synonym to case-based reasoning, to describe the typical case-based approach... However, it is also often used to characterize methods that solve new problems based on past cases from a different domain, while typical case-based methods focus on indexing and matching strategies for single-domain cases."

- Expert System is based on analogical structure
- Applying known solution to unknown problems.
- Example:
  Product of odd numbers is odd-Known
  Find Product of even numbers?