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.**

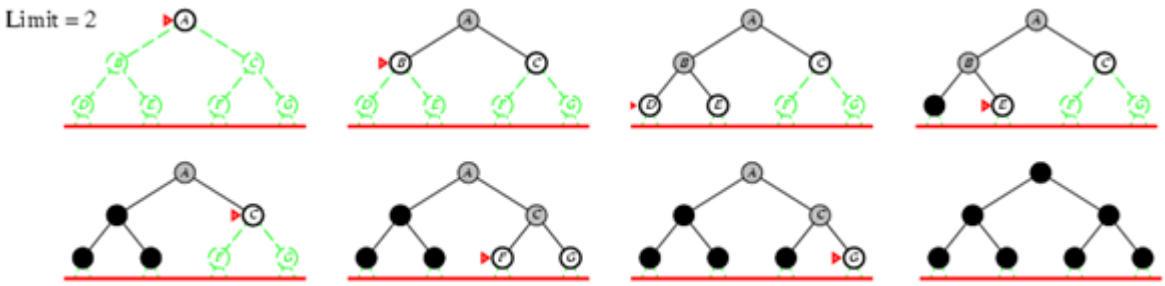
To avoid the infinite depth problem of DFS, we can decide to only search until depth L, i.e. we don't expand beyond depth L. This is called Depth-Limited Search. Suppose if the solution is deeper than L, then increase L iteratively. This is known as Iterative Deepening Search. This iterative deepening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

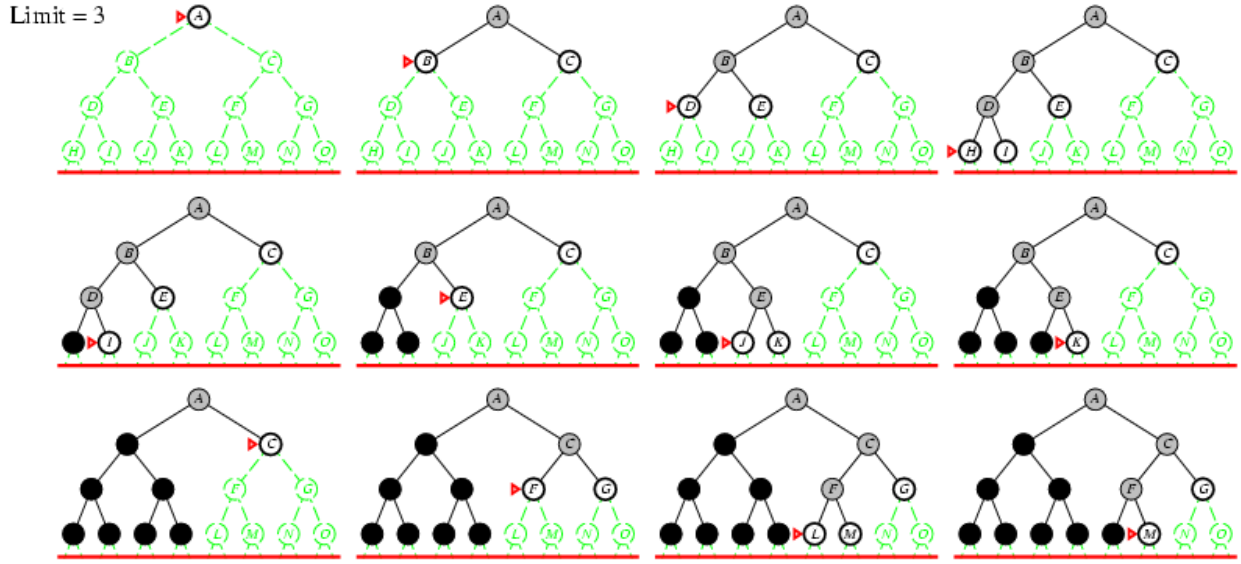
- For a branching factor b of 2, this might double the search time.
- For a branching factor b of 10, this might add 10% to the search time.

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

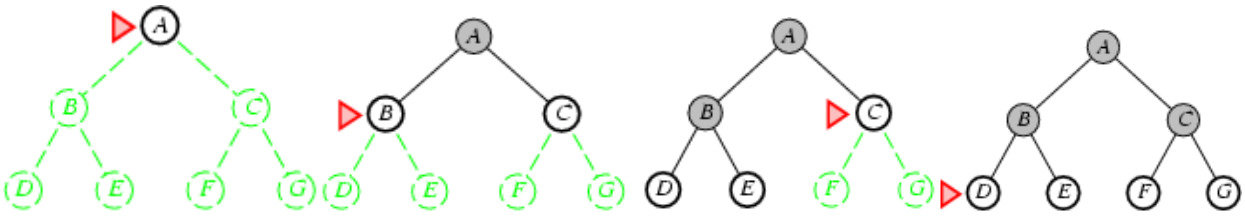
Space Complexity - $O(bd)$

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,

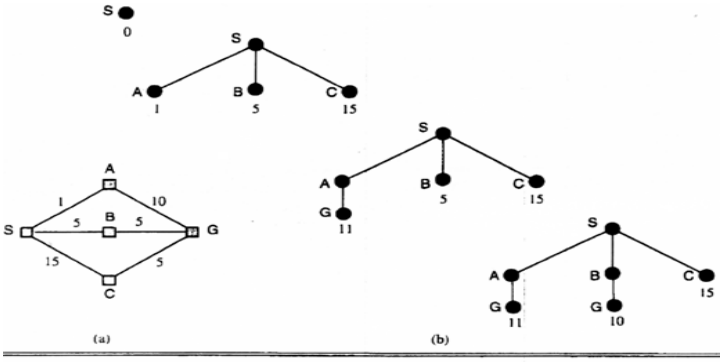
- either in fringe or on a path to fringe).

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

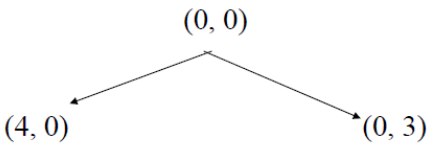
If we choose another control strategy, let us say, choose a rule randomly from the applicable rules then definitely it causes motion and eventually will lead to a solution. But one may arrive to same state several times. This is because control strategy is not systematic.

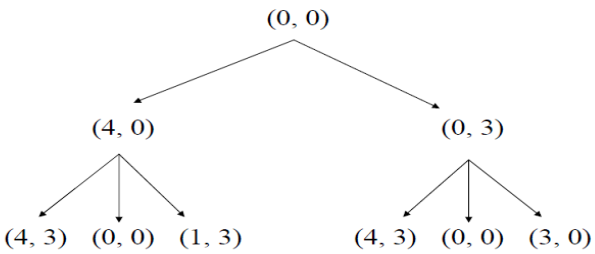
Systematic Control Strategies (Blind searches)

i) Breadth First Search

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

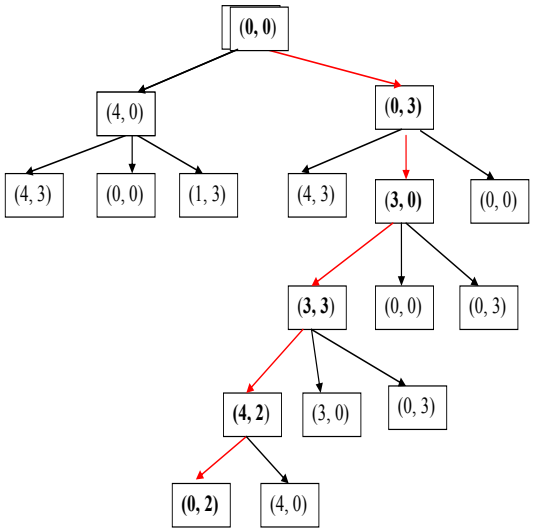
•Construct a tree with the initial state as its root.
•Generate all the offspring of the root by applying each of the applicable rules to the initial state.
•Now for each leaf node, generate all its successors by applying all the rules that are appropriate
•Continue this process until some rule produces 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.

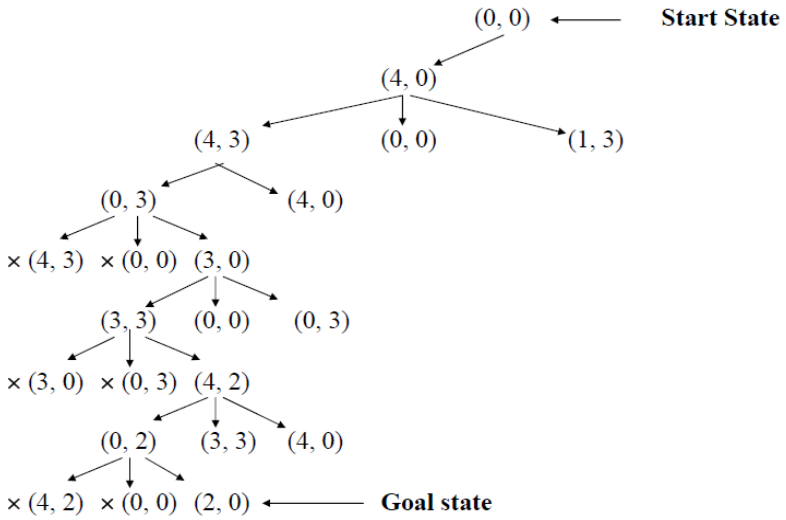
ii) Depth First Search:

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.



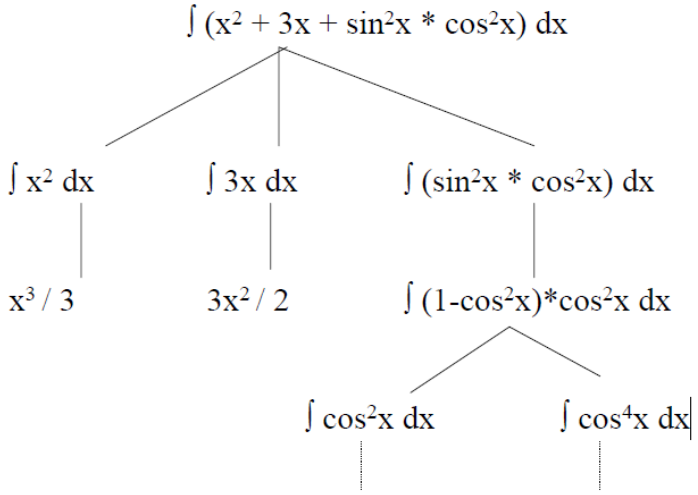
3. **Explain how different problem characteristics are analyzed in detail.**

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

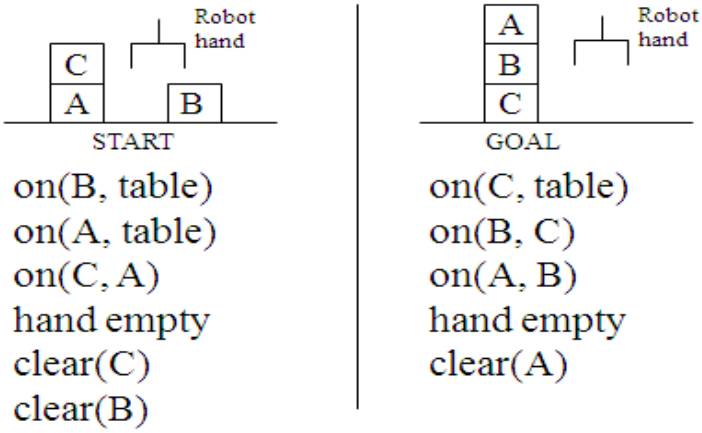
Decomposable problems can be solved by the divide-and-conquer technique. In divide and conquer technique, divide the problem in to sub-problems, find the solutions for the sub-problems. Finally, integrate the solutions for the sub-problems, we will get the solution for the original problem. Suppose we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

2	**8**	**3**
1	**6**	**4**
7		**5**

8		**4**
7	**6**	**5**

\

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

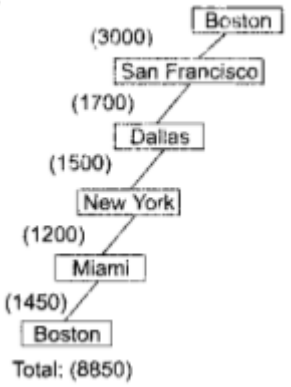
7. It is now 1991 AD.	**7**
5. All Pompeian died in 79 AD.	**5**
11. All Pompeian are dead now.	**7, 5**
2. Marcus was a Pompeian.	**2**
10. Marcus was dead	**11, 2**

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.

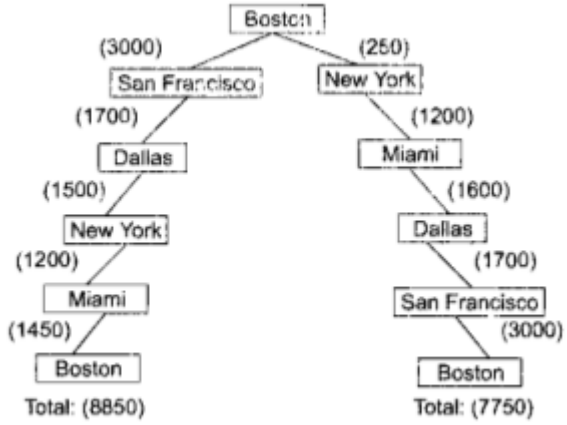
Best path problem

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.

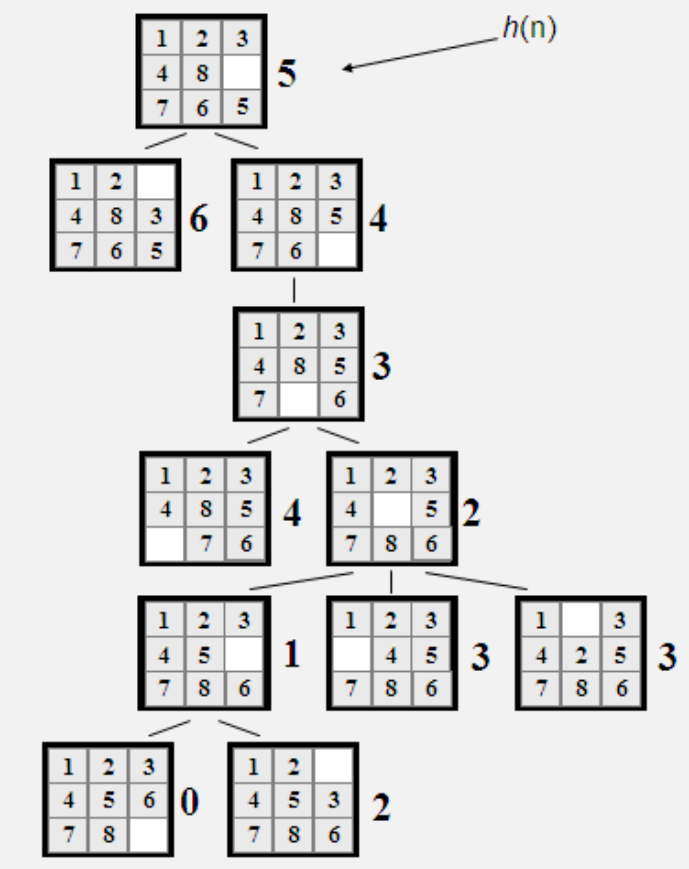
Heuristics might underestimate or overestimate the merit of a state. But for reasons which we will see, heuristics that only underestimate are very desirable, and are called admissible.

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.

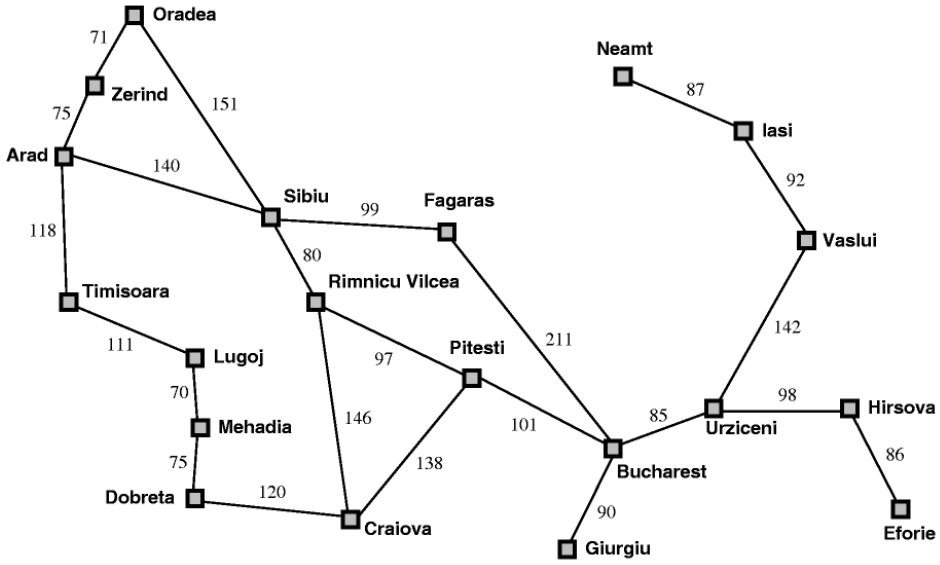
- Backtrack to some earlier node and try going in a different direction. This is a good way of dealing with local maxima.
- Make a big jump to try to get in a new section of the search space. This is particularly a good way of dealing with plateaus.
- Apply two or more rules before doing a test. This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

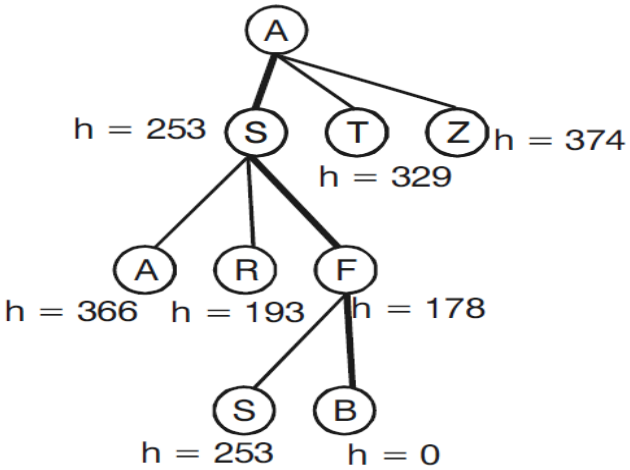
(a) Pick the best node on OPEN

(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

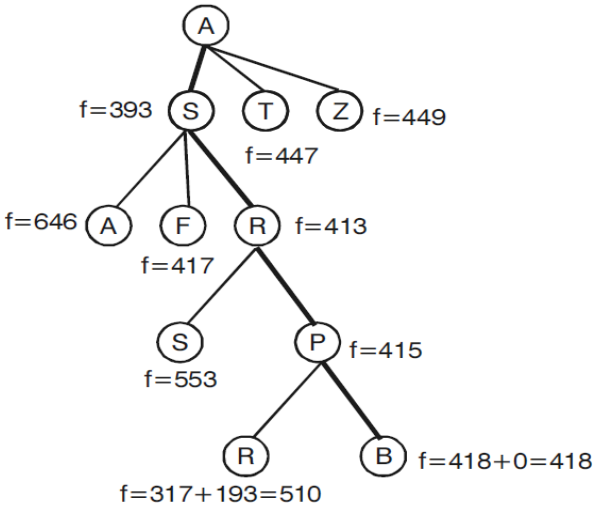
Evaluation function

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

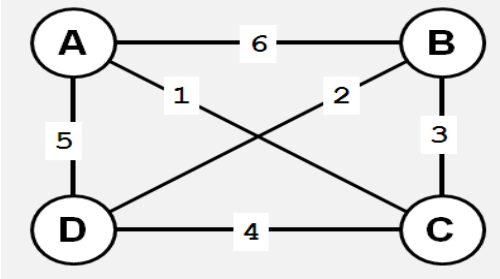
Algorithm

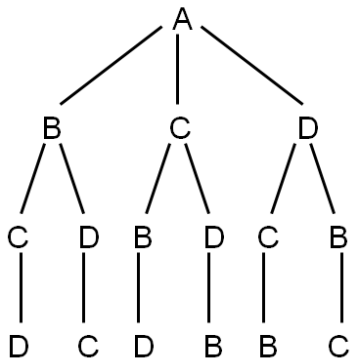
1. Generate a possible solution. For some problems, this means generating a particular point in the problem space. For others, it means generating a path from a start state.
2. Test to see if this is actually a solution by comparing the chosen point or the endpoint of the chosen path to the set of acceptable 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.





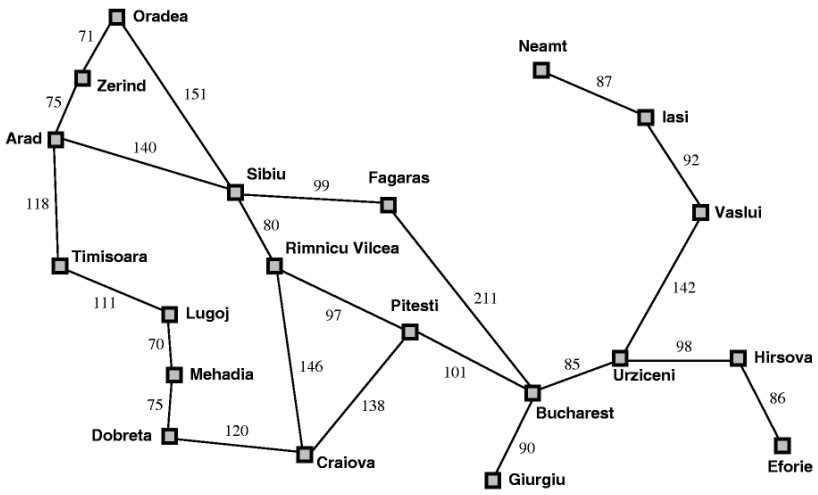
- TSP - generation of possible solutions is done in lexicographical order of cities:
  1. A - B - C - D
  2. A - B - D - C
  3. A - C - B - D
  4. A - C - D - B
  5. A - D - C - B
  6. A - D - B - C
- *n=80* will take millions of years to solve exhaustively!

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

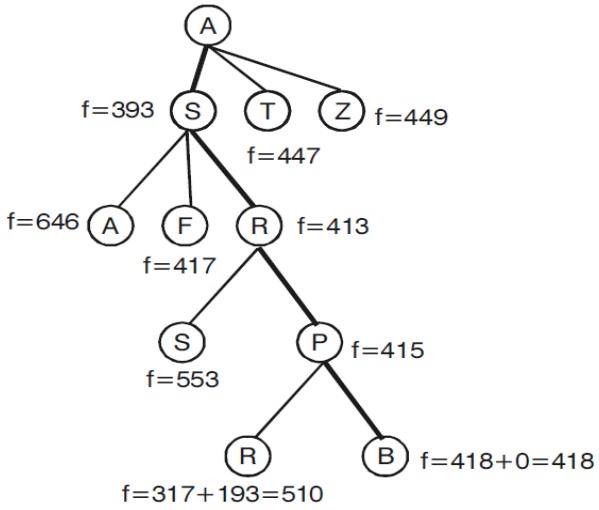


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

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

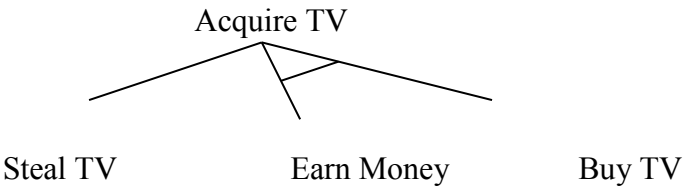
(LAST-PART←MEA(O-RESULT, GOAL))

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.

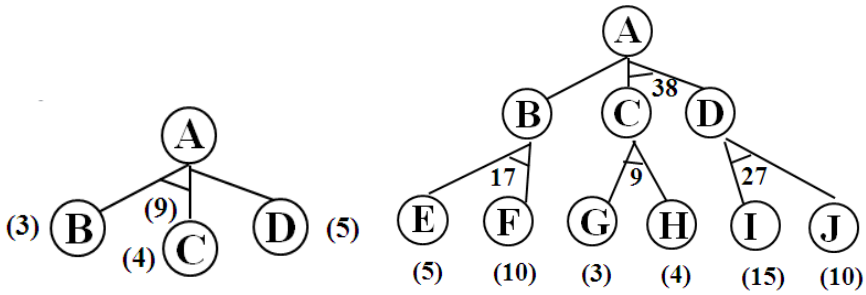


To find a solution in AND–OR graph, we need an algorithm similar to A* with the ability to handle AND arc appropriately.

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.

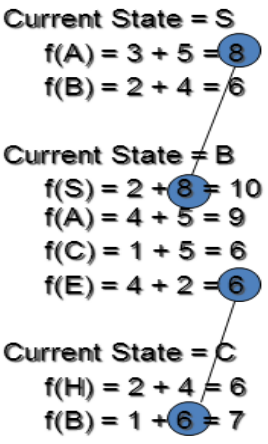
   c) Change the f' estimate of the newly expanded need to reflect the new information provided by its successors. Propagate this change backward through the graph. If any node contains the successor are whose descendants are all solved, label the node itself as SOLVED. At each that us visited while going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

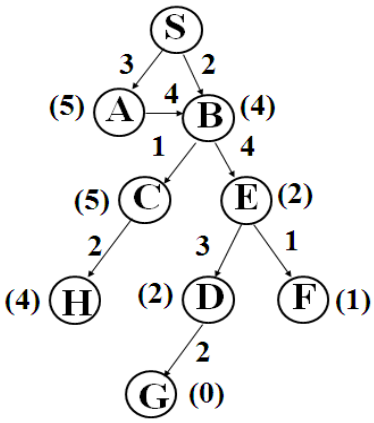
arcs is the most promising and mark it as part of the current best path. This may cause the current best path to change. This prorogation of revised cost estimates back up the tree was not necessary in the best first search algorithm.

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.



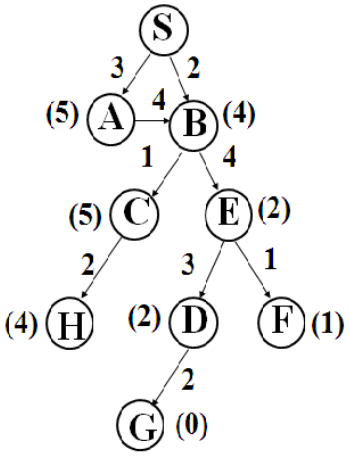


Current State = H
f(C) = 2 + 7 = 9

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



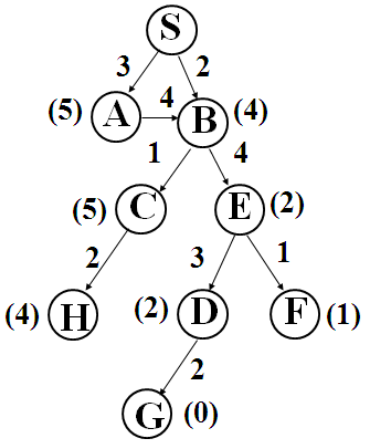
Current State = F
f(E) = 1 + 5 = 6

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:

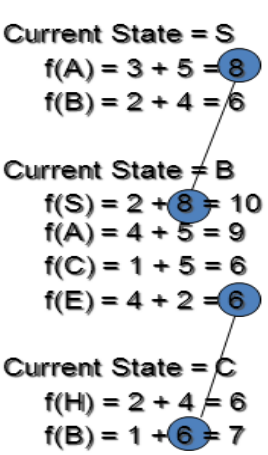
i. Remove a node from S and call it CURRENT.

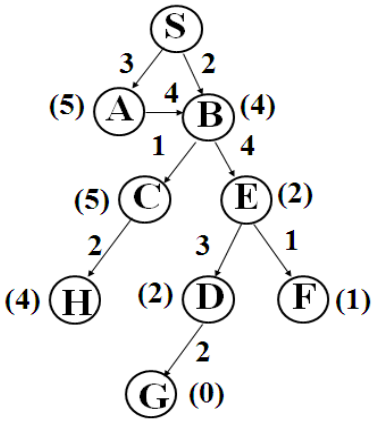
ii. Compute the cost of each of the arcs emerging from CURRENT. Assign minimum cost of its successors as its h.

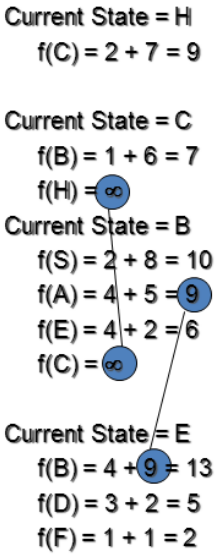
iii. Mark the best path out of CURRENT by marking the arc that had the minimum cost in step ii

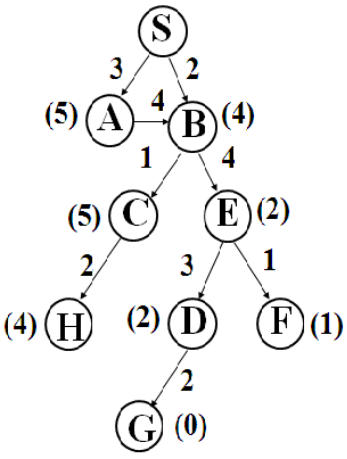
iv. Mark CURRENT as SOLVED if all of the nodes connected to it through new labelled arc have been labelled SOLVED

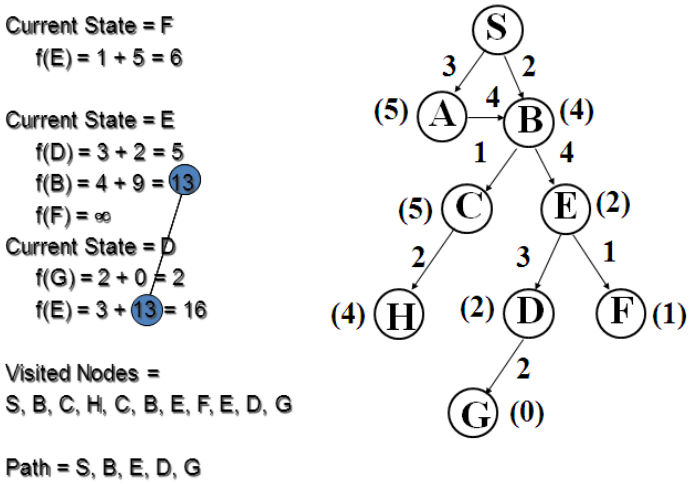
v. If CURRENT has been labelled SOLVED or its cost was just changed, propagate its new cost back up through the graph. So add all of the ancestors of CURRENT to S.











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)**
   ∀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.

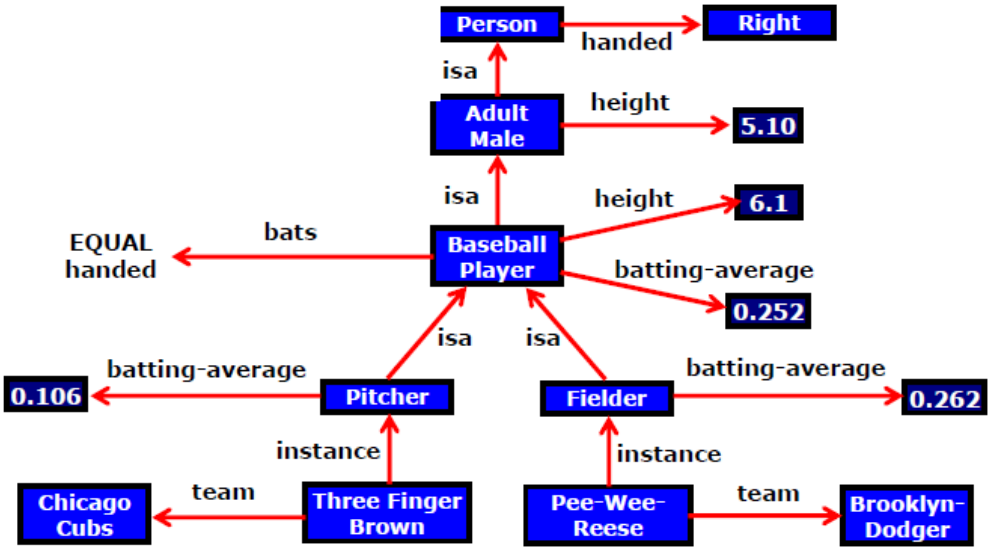
The table below shows a simple way to store facts. The facts about the set of objects are put systematically in columns.

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)" or "Something is blue" as "∃x Blue(x)".	**Example** Everything is green" as "G(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

∀x(dog(x) ∧barking(x)→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. $\forall w\ \neg\ smile(w)$

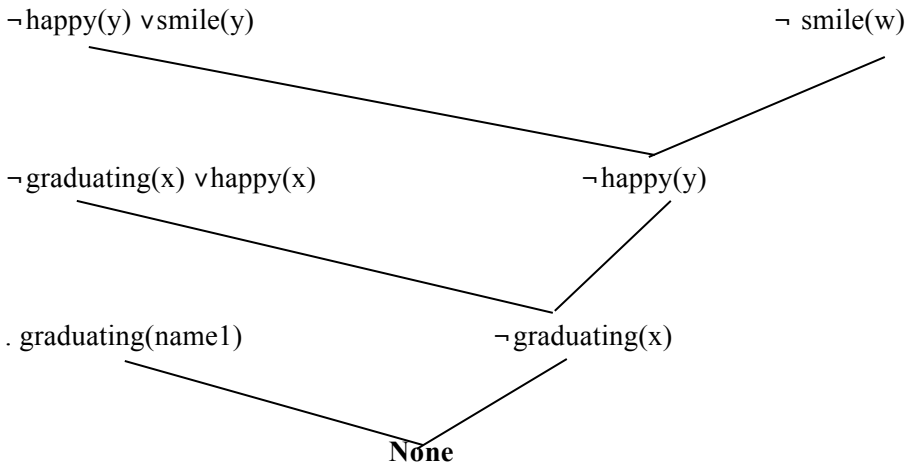
(v) Eliminate ∃

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. $\neg\ smile(w)$

(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)

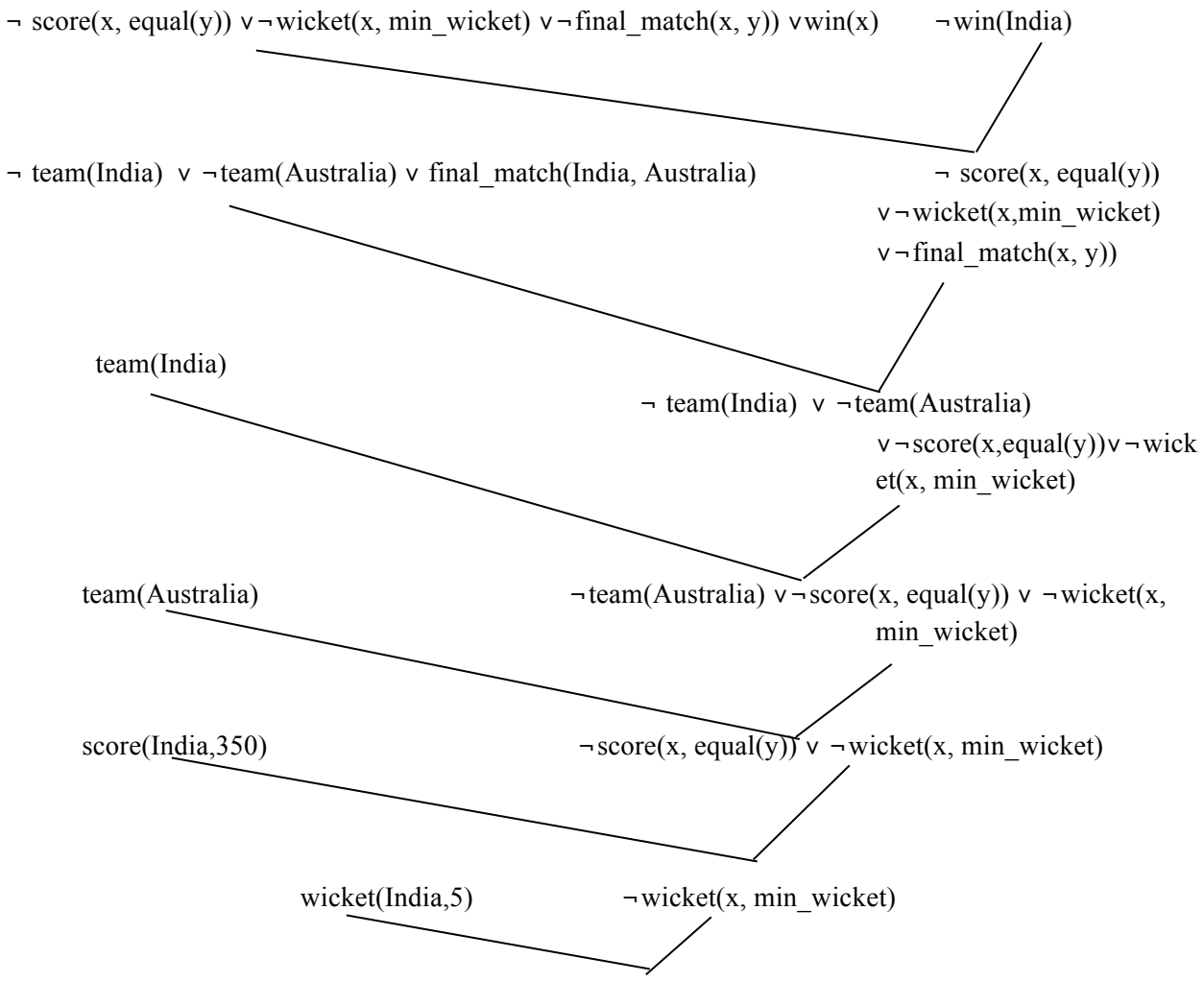
(e) ¬ team(x) ∨ ¬wins(x) ∨score(x, max_runs))

(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)



None

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)

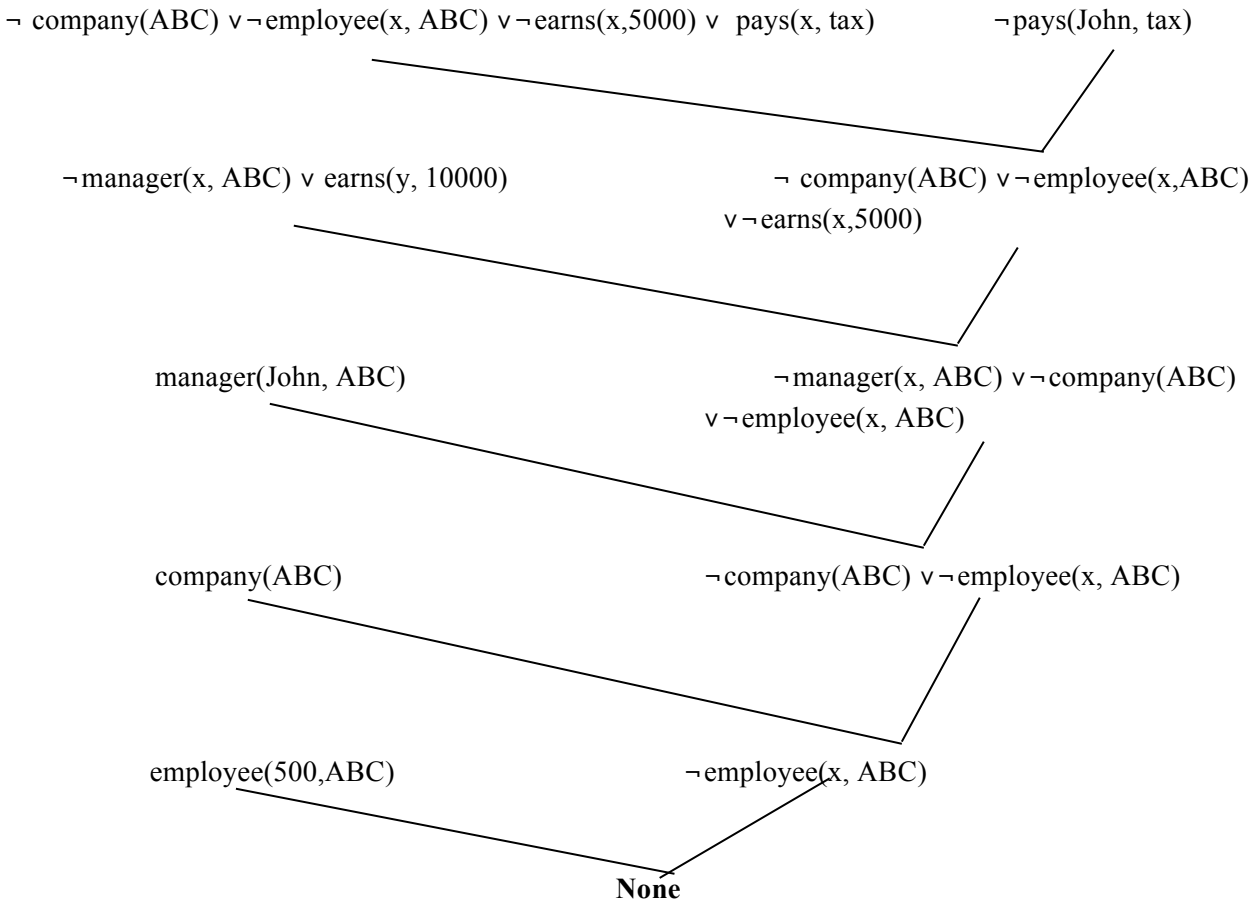
3. manager(John, ABC)

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)

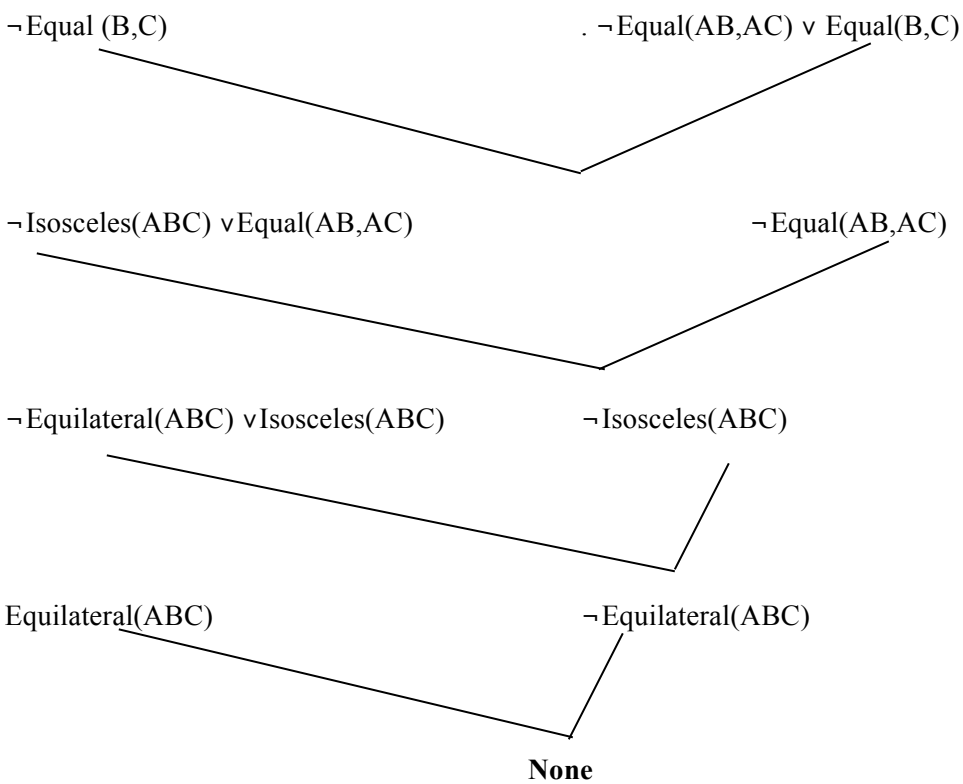
Convert to clausal form

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.

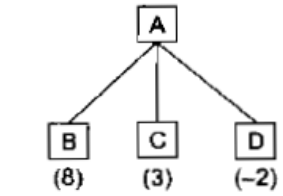
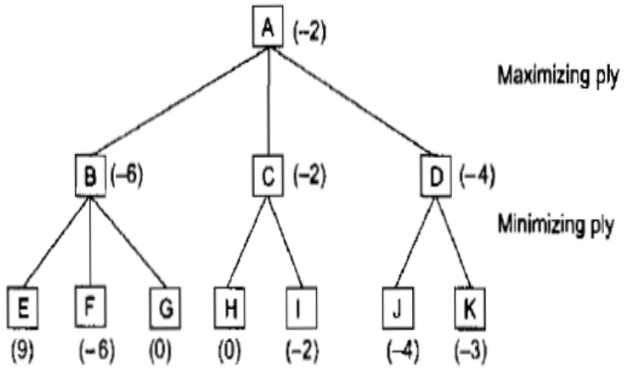


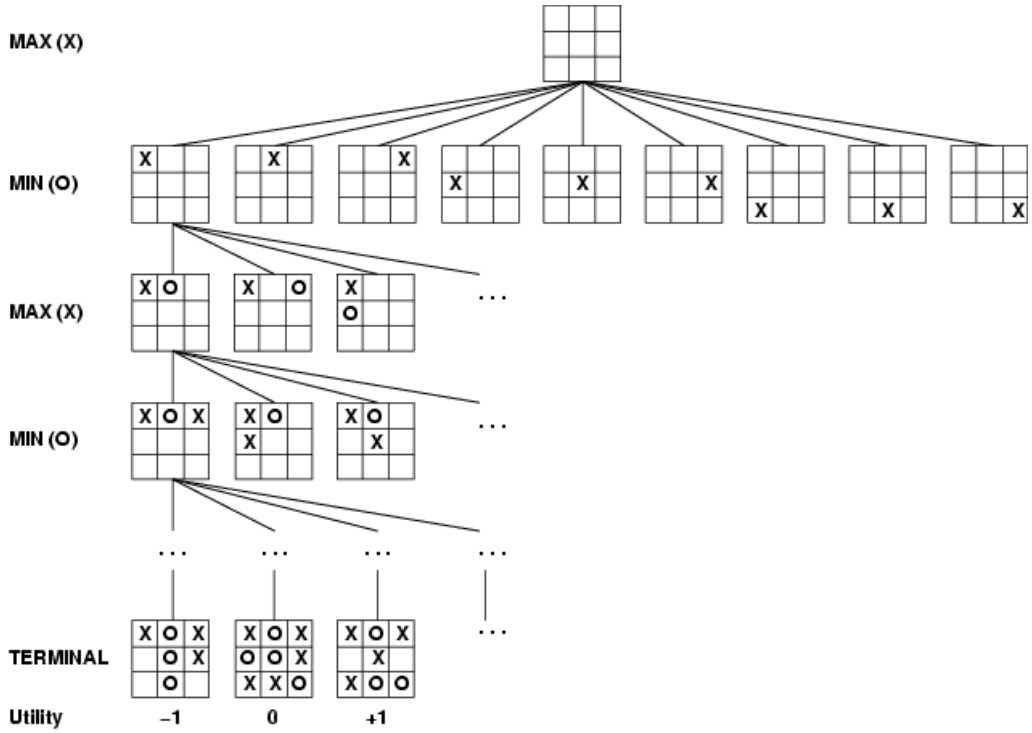
Fig. 12.1 *One-Ply Search*

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

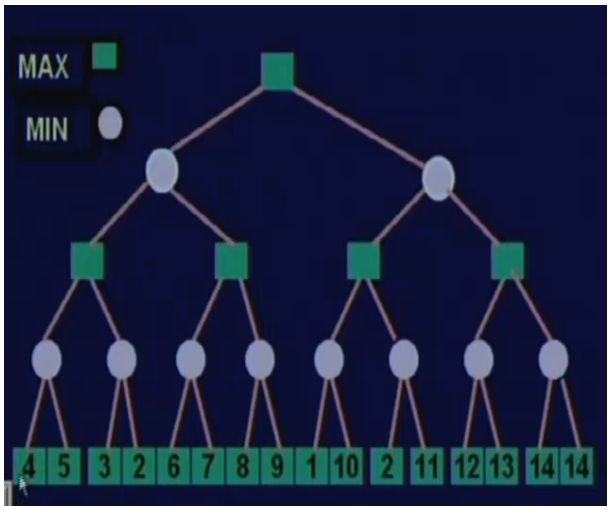
If the maximum depth of the tree is m, and there are b legal moves at each point, then the time complexity of the minimax algorithm is O (b^m).

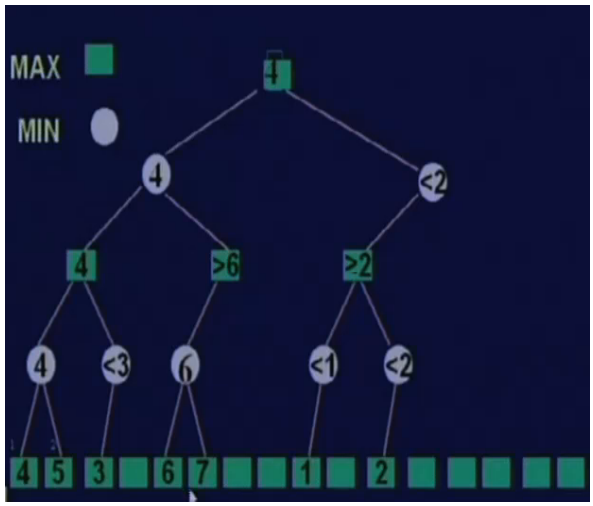
9. Explain alpha-beta pruning in detail along with example.

Alpha Beta Cut-off

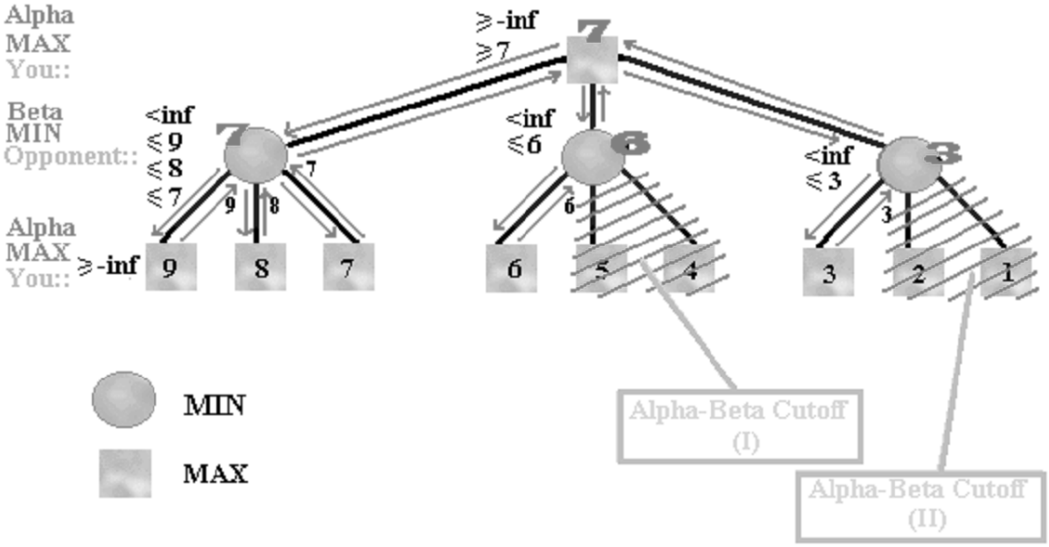
MINIMAX searches the entire tree, even if in some cases the rest can be ignored. But, this alpha beta cutoff returns appropriate minimax decision without exploring entire tree. The minimax search procedure is slightly modified by including branch and bound strategy one for each players. This modified 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.

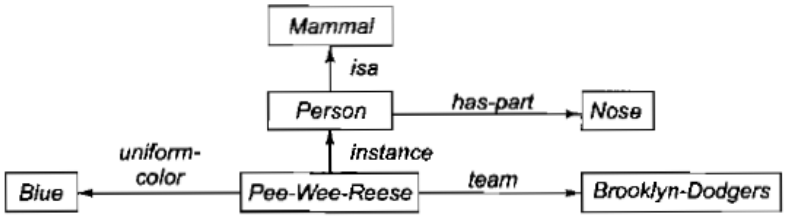
Modeling-based representations reflect the structure of the domain, and then reason based on the model.

- **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

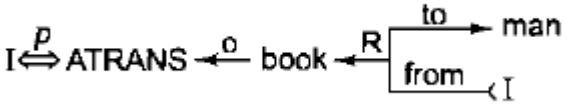
isa	:Team
cardinality	:26
team-size	:24

(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.
2. How does an inference engine work in a frame based system?
3. Explain the need of fuzzy set and fuzzy logic with example. (MAY/JUNE 2012 & NOV/DEC 2013)
4. Explain the method of performing exact inference in Bayesian networks. (NOV/DEC 2012)
5. Explain in detail about forward and backward chaining with suitable example.
6. Explain knowledge representation in detail with example.
7. Explain Rule based system with example.
8. Explain Dempster - Shafer theory with example.
9. i. Write the enumeration algorithm for answering queries on Bayesian networks. (8)
   ii. Describe a method for constructing Bayesian networks. (8) (MAY/JUNE 2013)
10. Write notes on:
    i) Fuzzy reasoning
    ii) Bayesian probability
    iii) Certainty factors

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.

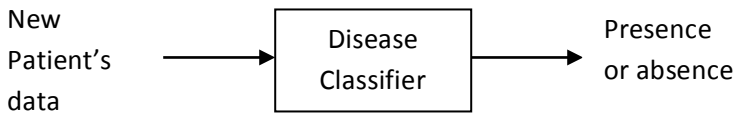
A computer program is said to learn from experience E with respect to some class of tasks T and performance P, if its performance at the tasks improves with the experiences.

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

.... Training



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 STRIPES mechanism with example.
2. Explain the various components of a Planning system.
3. What do you mean by plan generation? Explain Basic plan generation system with an example.
4. Briefly explain the advanced plan generation system.
5. Distinguish:
   i) Production based system
   ii) Frame based system
6. Explain the concept of planning with the Block world example.
7. Discuss simple planning using a Goal Stack.
8. Write briefly the concept of Learning.
9. Explain Machine learning and Adaptive learning with example.
10. Solve the blocks world problem using strips. How does it act as a planning system?

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

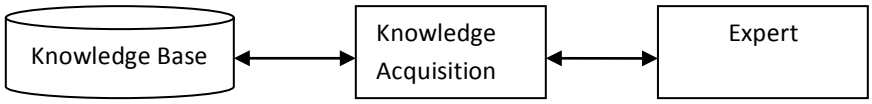
- Hybrid systems
- Model based systems
- Ready-made systems
- Real-time expert systems

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.
2. Elaborately explain the process of knowledge acquisition.
3. i) Explain the various stages of Expert system development.
   ii) Explain heuristics with a example
4. Draw the schematic diagram of an expert system. Explain all the relevant components.
5. i) Explain the components of expert systems with a neat diagram.

ii) Discuss the features of Expert systems

6. Discuss the advantages and limitations of expert systems.
7. i) Explain briefly about Meta knowledge
   ii) Explain the role of expert system
8. Write short notes on:
   a. MYCIN and its applications
   b. DART and its applications
9. Write notes on:
   i) Expert system shell
   ii) Limitations of Expert systems
10. Explain the pitfalls in selecting an expert system.