Artificial Intelligence

UNIT I and UNIT III

Definition of Artificial Intelligence (AI)

- Artificial Intelligence is the study of how to make computer do things which at the moment people do better.

- These are the approaches that gave AI the direction to evolve:

➢ Acting Humanly – The Turing Test approach

➢ Thinking Humanly – The cognitive modelling approach

➢ Thinking Rationally – The laws of thought approach

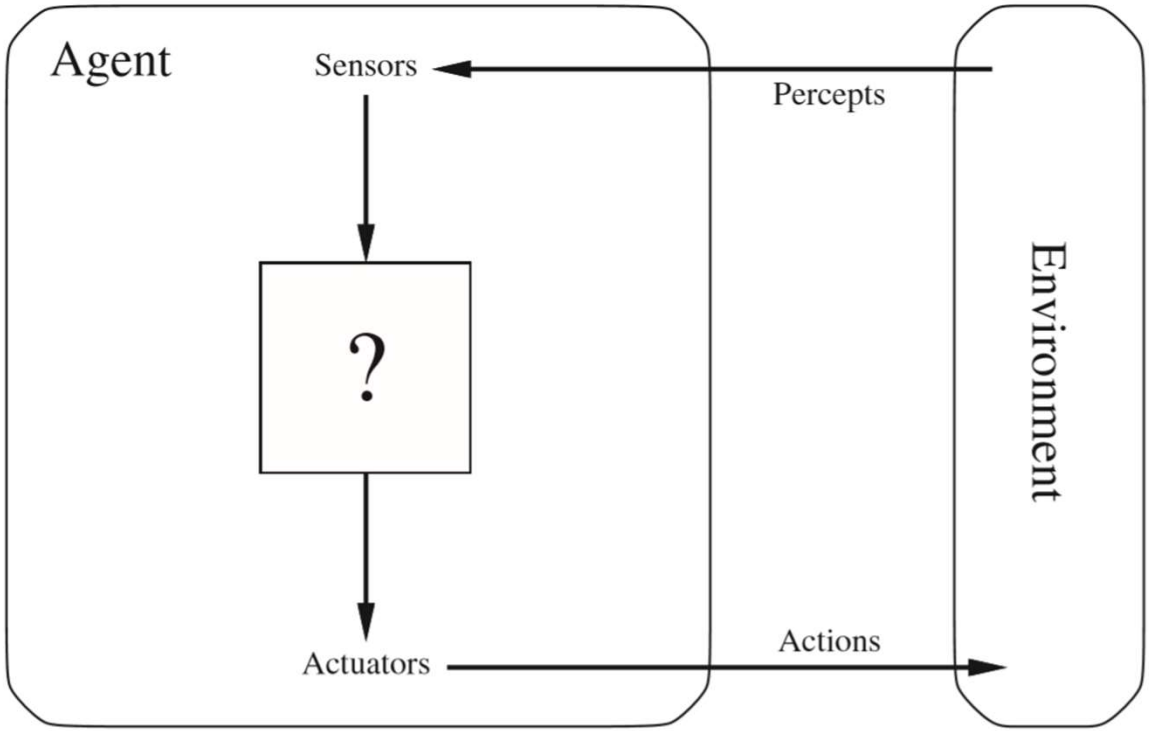
➢ Acting Rationally – The rational agent approach

Intelligent Agent

- An agent is something that perceive its environment through sensors and then respond through actuators.

- Percept sequence - It is the complete history of everything that an agent has ever perceived. Percept sequence leads to action.
- Agent function – It is the mapping of percept sequence to an action.
- Performance measure – It is the criteria that determines how successful an agent is.

Intelligent Agent



Rational Agent

- A rational agent should select an action that is expected to maximize its performance measure on basis of its percept sequence and its built in knowledge.

- Rationality depends on 4 things:

➢ The performance measure

➢ The agent's prior knowledge of the environment

➢ The actions that agent can carry out

➢ The agent's percept sequence

- Omniscience – An omniscient agent knows the actual outcome of its actions and can act accordingly but omniscience is impossible in reality.

- Learning – A rational agent should not only gather information but also learns from what it perceives.

- Autonomy – An agent should not only depend on built in knowledge by its designer but should also rely on its percepts.

Properties of Environment

- **Fully observable** – When agent's sensors can access the complete state of environment
- **Partially observable** – When agent's sensors can access only some parts of the state because of noisy or inaccurate sensors. Ex- A vacuum cleaner with only local dirt sensor cannot tell dirt in other squares.

- **Single agent** – Only one agent in an environment. Ex- Solving crossword puzzle require only one agent.
- **Multi agent** – More than one agent in an environment. Ex- Playing chess

Properties of Environment

- **Deterministic** – If next state is completely determined by the current state and action of agent. Ex- crossword puzzle.
- **Stochastic** – If next state depends on some probability. Ex- Taxi driving.

- **Episodic** – Agent's experience is divided into atomic episodes. Agent's current decision is not dependent on previous decision. Ex- To spot defective parts on assembly line.
- **Sequential** – The current decision could affect all future decisions. Ex- chess and taxi driving.

Properties of Environment

- **Static** – If the environment does not change while an agent is acting. Ex- Crossword puzzle.
- **Dynamic** – If the environment change during agent's actions or decision making. Ex- taxi driving.

- **Discrete** – If there are limited number of distinct states of the environment. Ex- Chess.
- **Continuous** – If state percept are continuously changing. Ex- Taxi driving.

Properties of Environment

- **Known** – If it has knowledge of the environment. Ex- solitaire card game (know rules but unable to see cards or the state).
- **Unknown** – If it does not have knowledge of the environment. Ex- video game (screen show entire state but don't know what buttons do i.e. the rules).

- The hardest case for an agent is partially observable, multiagent, stochastic, sequential, dynamic, continuous and unknown.

Structure of Intelligent Agents

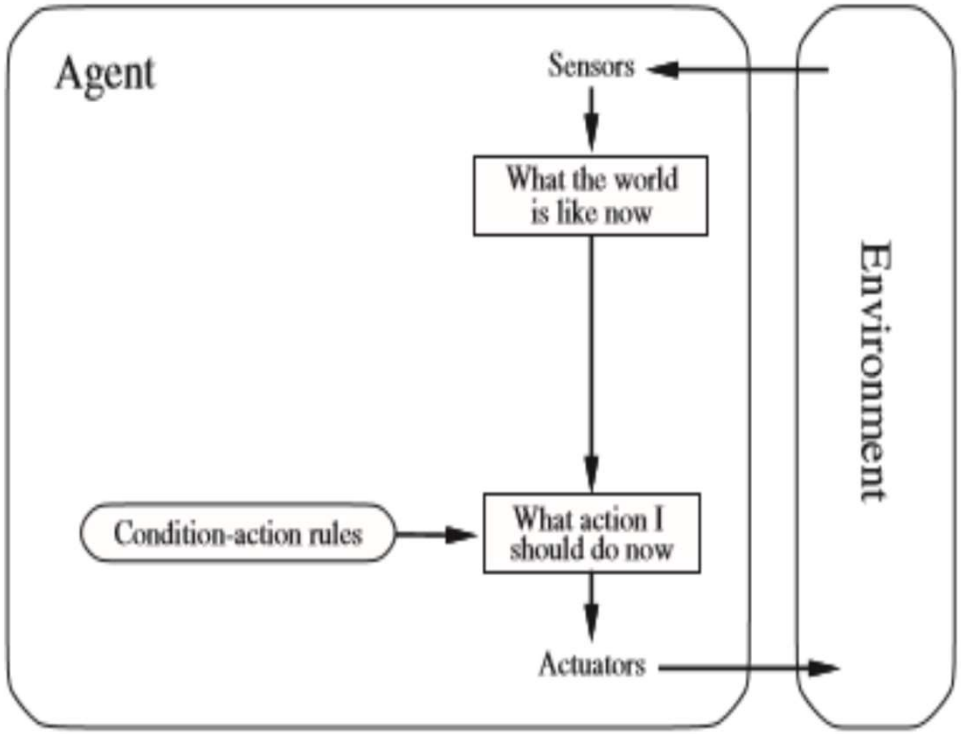
- Agent = architecture + program

- Architecture – Machinery or computing device on which an agent program executes. A program should be appropriate for the architecture. If program has action walk then architecture should have legs to perform it.

- Types of agents:
  - Simple reflex agents
  - Model based reflex agents
  - Goal based agents
  - Utility based agents

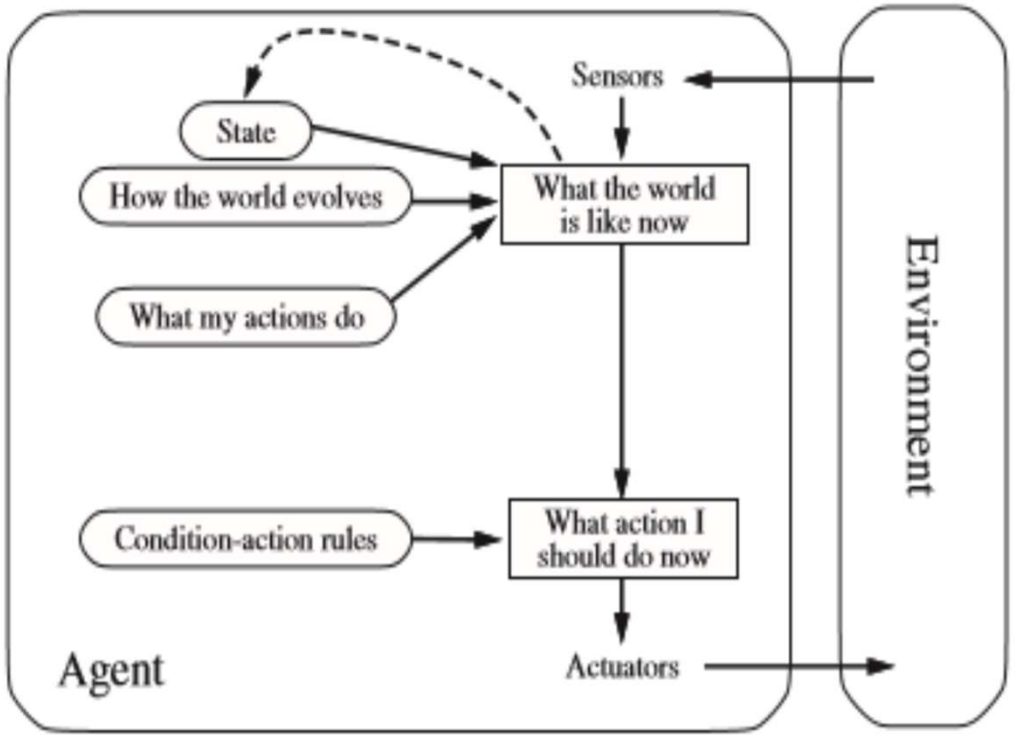
Simple reflex agents

- They respond directly to percept.
- They select actions on the basis of current percept, ignoring rest of the percept history.
- Their environment is completely observable.
- Condition-action rule maps a state(condition) to an action.
- These agents are simple but have limited knowledge.
- Ex- if car in front brakes then initiate braking



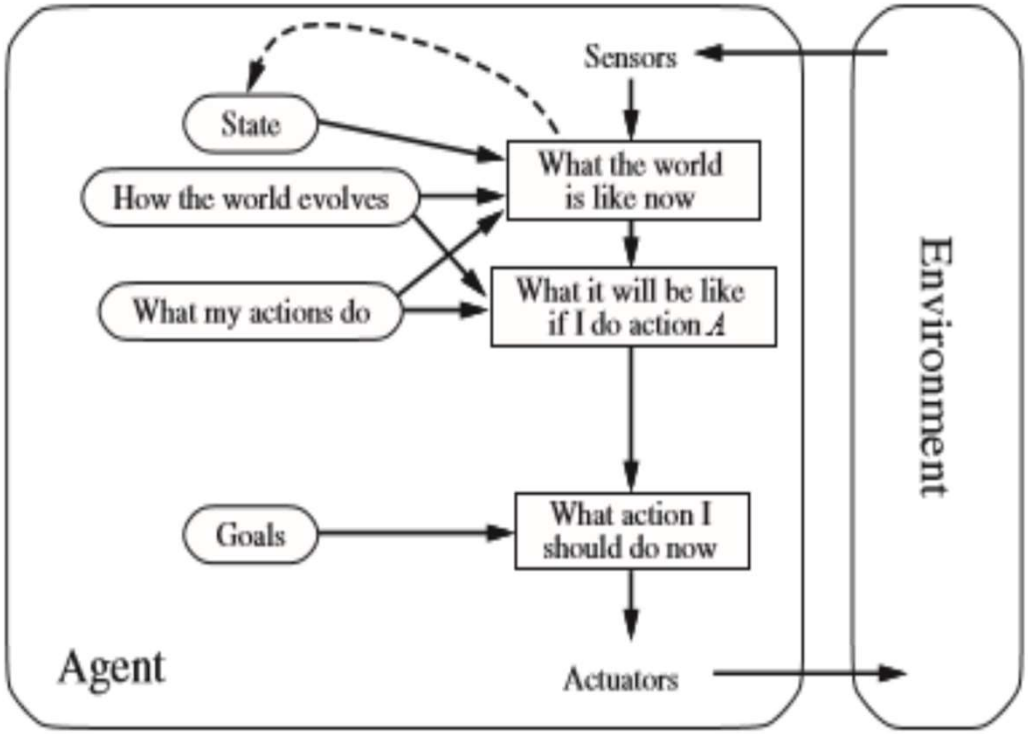
Model based reflex agents

- They do not respond directly but maintain internal state to track aspects of the world that are not evident in current percept.
- It maintains an internal state to keep track of the part of the world it cannot see now.
- It keeps knowledge about how the things happen in the world.
- Updating the state requires information about how the world evolves and how the agent's actions affect the world.
- Ex- In brake light of older car model, it will be difficult for agent to tell if the car in front is braking or turning.



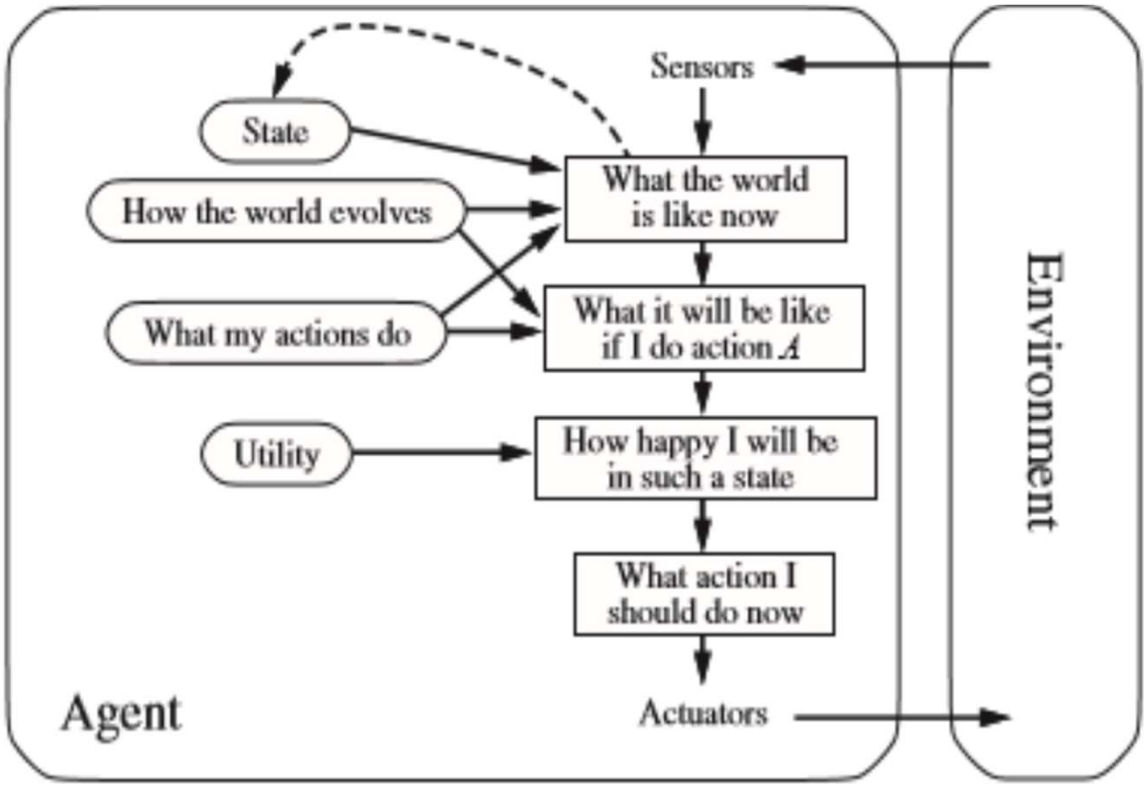
Goal based agents

- They act or choose their actions to achieve goal.
- It keeps track of the world state as well as a set of goals that it is trying to achieve and it will choose an action that will lead to the achievement of its goal.
- Ex- At road junction a taxi can turn left or right or go straight but correct decision depends on where the taxi is trying to get to.
- Ex- In case of braking reflex agent will brake but goal based agent will slow down.



Utility based agents

- They try to maximize their own expected happiness.
- They choose actions based on a preference (utility) for each state.
- It has advantage in terms of flexibility and learning.
- There are conflicting goals, out of which only few can be achieved. Ex- between speed and safety.



Problem Solving

Problem solving agent is a kind of goal based agent. They decide what to do by finding sequences of actions that will lead to the desirable states.

- Goal formulation – Is the first step in problem solving of creating goal.
- Problem formulation – It is the process of deciding which actions or states to consider, given a certain goal.
- Search – The process of looking for a sequence of actions that can lead to states of known value, then choosing the best one. We have search algorithm the input for which will be a problem and solution will be action sequence.
- Execute – In this phase action recommended by the algorithm is carried out.

Well Defined Problems

A problem can be well defined by five components:

- Initial State – In which the agent is in, initially.
- State Space – All available states that can be reached from the initial state by any sequence of actions.
- Goal state – The state that an agent wants to get to.
- Operators – It describes action, like which state will be reached by that action.
- Path – It is any sequence of action leading from one state to another.

Control Strategy

It specifies the order in which the operators will be compared and a way of solving the conflicts that arise when several rules matches at once.

A good control strategy

- creates motion of agent in the state
- And is systematic

Problem Representation

Most common methods of problem representations in AI are:

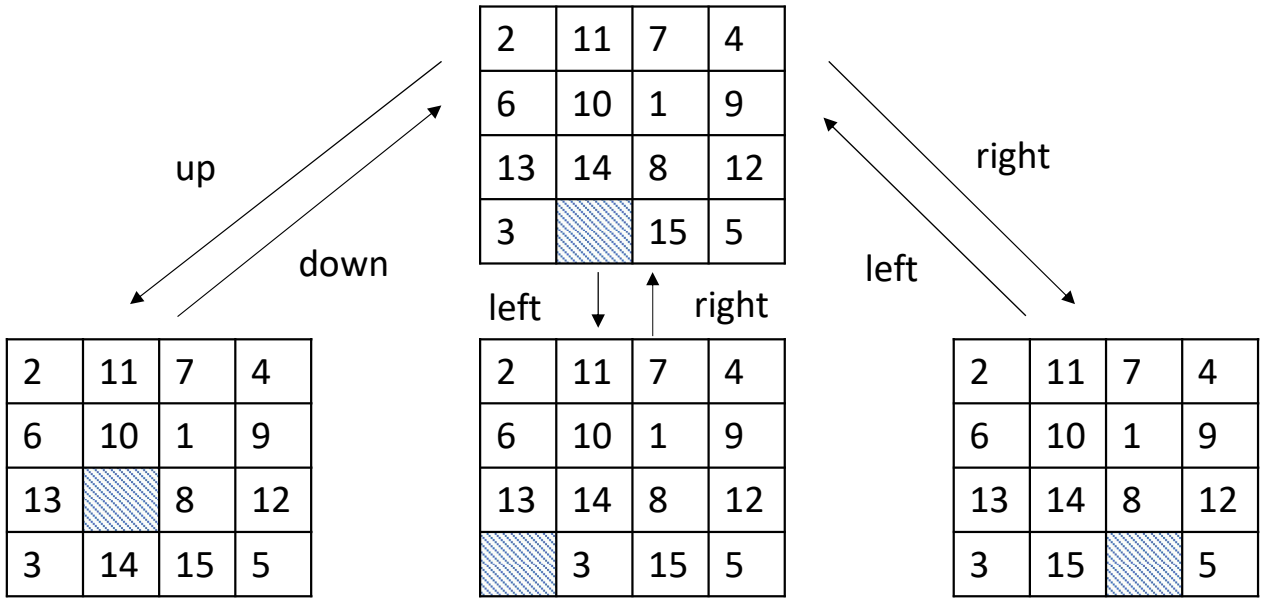
- State Space Representation
- Problem Reduction

State Space Representation

- A set of all possible states for a given problem is known as the state space of the problem. State space representation are highly beneficial in AI because they provide all possible states, operations and the goals.
- If the entire state space representation for a problem is given, it is possible to trace the path from the initial state to the goal state and identify the sequence of operators necessary for doing it.
- Major deficiency of this method is that it is not possible to visualize all states for a given problem and the resources of the computer system are limited to handle huge state space representations.

Example state space representation

Fragment of the state space of 15 tile puzzle

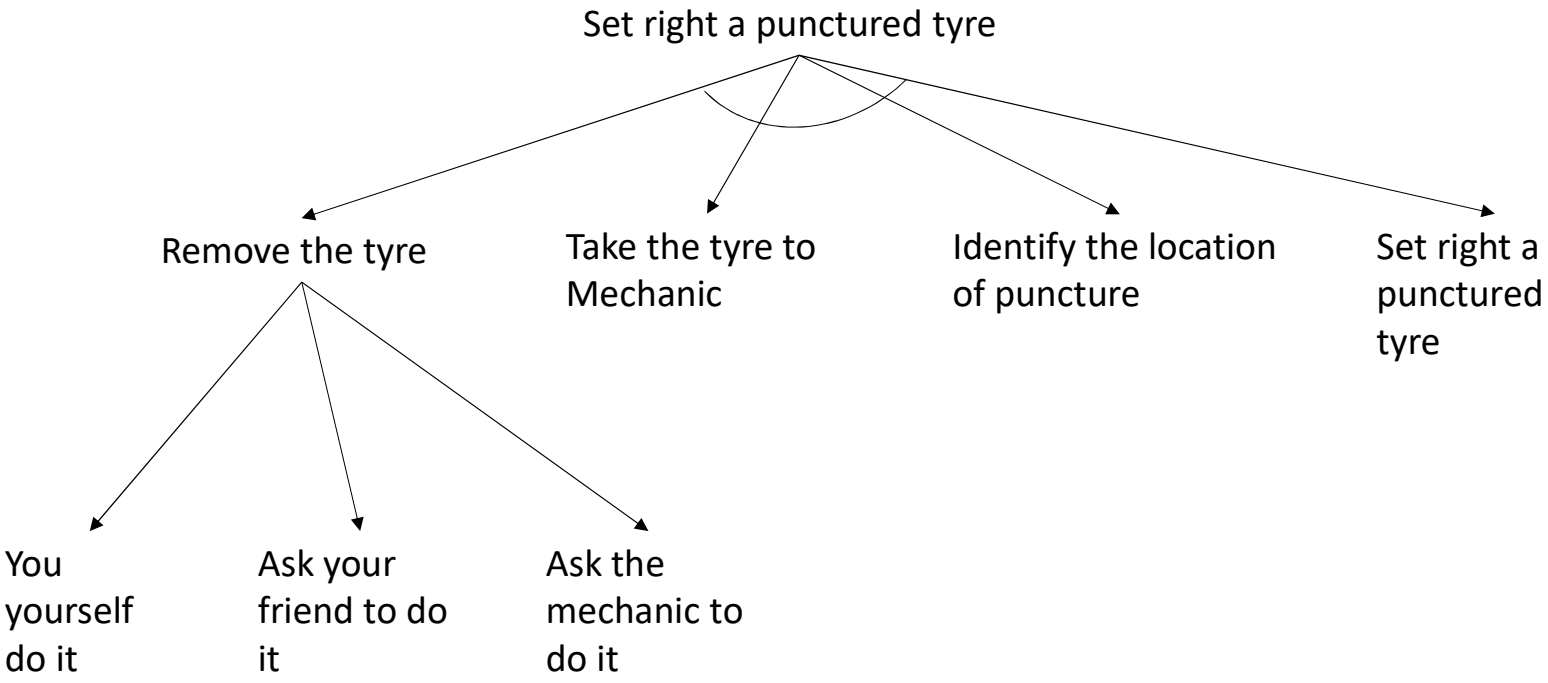


Problem Reduction

- To reduce the deficiencies of the state space representation method, problem reduction method becomes useful.

- In this method a complex problem is broken down or decomposed into a set of primitive sub problems. Solutions for these primitive sub problems are easily obtained.

Example of Problem Reduction



Water Jug Problem

- You are given 2 jugs, a 4 gallon one and a 3 gallon one. Neither has any measuring marks on it. There is a pump that can be used to fill the jugs with water. How can you get exactly 2 gallon water in the 4 gallon jug?

- State space – Set of ordered pairs of integers (x,y), x represents number of gallons in 4 gallon jug (x = 0,1,2,3,4) and y represents number of gallons in 3 gallon jug (y = 0,1,2,3).

- Start state/initial state – (0,0) i.e. both jugs are empty
- Goal state – (2,y) for any value of y

Water Jug Problem

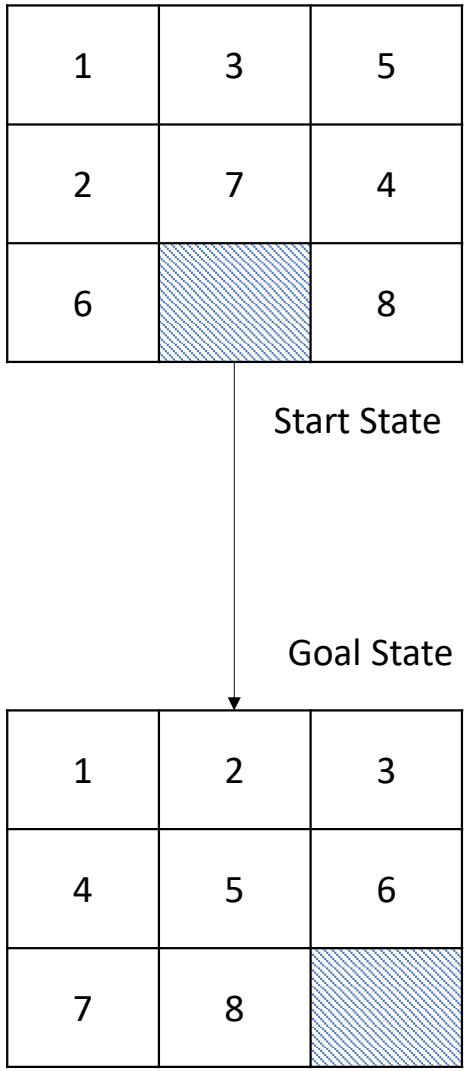
Operators/ actions	**State**	**Next state**
1. Fill 4 gallon jug	If x<4	(4,y)
2. Fill 3 gallon jug	If y<3	(x,3)
3. Empty 4 gallon jug	If x>0	(0,y)
4. Empty 3 gallon jug	If y>0	(x,0)
5. Pour water from 3 gallon jug to fill 4 gallon jug entirely	If (x+y>=4) and y>0	(4,y-(4-x))
6. Pour water from 4 gallon jug to fill 3 gallon jug entirely	If (x+y>=3) and x>0	(x-(3-y),3)
7. Pour all the water from 3 gallon jug into 4 gallon jug	If (x+y<=4) and y>=0	(x+y,0)
8. Pour all the water from 4 gallon jug into 3 gallon jug	If (x+y<=3) and x>=0	(0,x+y)

Water Jug Problem

Gallons in 4 gallon jug (x)	Gallons in 3 gallon jug (y)	Operators applied
0	0	Start state
4	0	By applying 1
1	3	By applying 6
1	0	By applying 4
0	1	By applying 8
4	1	By applying 1
2	3	By applying 6
2	3	Goal state

8 Puzzle Problem

- It is a puzzle where we have grid having 9 spaces and tiles in each space is numbered from 1 to 8. We move by sliding the tile.
- State space – It is any arrangement of 1 to 8 numbered tile and a blank tile on board.
- Start state – The starting configuration given.
- Operators – Four operators for moving blank tile left, right, up and down.
- Goal state – tiles arranged according to sequence from 1 to 8.
- A variant of 8 puzzle is 15 puzzle having grid with 16 spaces.



The Search Process

After defining a problem we will see how to recognize a solution. To find a solution we will do it by searching through the state space.

- Searching can be defined as a sequence of steps that transforms the initial state to a goal state.
- Searching process in AI can be broadly classified into two major categories:

1. Uninformed search or brute force search
2. Informed search or heuristic search

Uninformed Search

It is also called blind search meaning they have no information about the number of steps or the path cost from the current state to the goal. All they can do it distinguish goal state from a non goal state.

Following algorithms make use of uninformed search:

- Breadth first search
- Depth first search

Informed (Heuristic) Search

Informed search strategies are one that uses problem specific knowledge and can find solutions more efficiently. They are also called **heuristic search**.

Heuristics – Heuristics are approximations used to minimize the searching process. By their use the process of searching can be greatly reduced. We use heuristic function h( ) which maps the problem states into numbers.

Following algorithms make use of informed search:

- Hill Climbing
- Steepest hill climbing
- Best first search
- A *
- AO *

Uninformed Search

- Breadth first search
- Depth first search

Breadth First Search (BFS)

- In this strategy the root node is expanded first then all the nodes generated by root node are expanded next and then so on. All nodes at the depth $1^{(d)}$ are expanded before the nodes at depth $2^{(d+1)}$. Because of its systematic strategy if there is a solution BFS is guaranteed to find it and if there are several solutions then BFS will always find the shallowest (i.e. situated at less depth) state first.

- Breadth first search is implemented using a **QUEUE (FIFO)**.

Depth First Search (DFS)

- It expands one of the nodes at the deepest level of the tree. When the search hits a dead end then the search goes back and expand nodes at the shallower levels. As for the memory requirement it needs to store only a single path from the root node to a leaf node.

- Depth first search is implemented using a **STACK**.

Time complexity of BFS and DFS

Given a state branching factor is a number of different next state that we can possibly have.

b: branching factor

d: Depth of the goal

m: Depth of the state space

Time required for BFS: $1 + b + b^2 + b^3 + \ldots\ldots\ldots\ldots\ldots + b^d = O(b^d)$

In BFS we are starting with a node at depth d then we must be having all the other nodes at depth d already in OPEN.

Space: $O(b^d)$

In DFS the depth of goal is not important because we are not going level by level rather we are going down in the state space.

Time required for DFS: $O(b^m)$

Space: O(bm) (this is less than BFS)

Informed (Heuristic) Search

- Hill Climbing
- Steepest hill climbing
- Best first search
- A *
- AO *

Hill Climbing

Hill climbing is a form of generate and test. In this feedback from the test procedure is used to help the generator to decide which direction to move in the search space.

Algorithm for hill climbing:

Step1: Put the initial node on a list START.

Step2: If(START is empty) or (START = GOAL) then terminate search.

Step3: Remove the first node from START. Call this node a.

Step4: If(a = GOAL) terminate search with success.

Step5: Else if node ‘a’ has successors, generate all of them. Find out how far they are from goal node. Sort them by the remaining distance from the goal and add them to the beginning of the START.

Step6: Goto Step2

Steepest Hill Climbing

It is a variant of simple hill climbing it considers all the moves from the current state and selects the best one as the next state. This search is also called as gradient search.

Algorithm for steepest hill climbing:

Step1: Put the initial node on a list START.

Step2: If(START is empty) or (START = GOAL) then terminate search.

Step3: Remove the first node from START. Call this node a.

Step4: If(a = GOAL) terminate search with success.

Step5: Else if node ‘a’ has successors, generate all of them. Assume a list as SUCC such that:

SUCC = highest-valued successor of current node ‘a’

If value(SUCC) < value(current) then set value(SUCC) = value(current)

Else set current = SUCC

Step6: Goto Step2

Steepest Hill Climbing

Working - In this we take an additional list as SUCC such that comparing with SUCC the next node will be either better than SUCC or not better than SUCC. If it is better than SUCC then we change SUCC to that node's value else if it is not better then SUCC becomes the current node.

Properties:

- Terminates when peak is reached
- Does not look ahead
- Does not backtrack

Problem in Hill and Steepest hill climbing

Problems in the hill and steepest hill climbing techniques:

- Local maximum – A state that is better than all its neighbours but not so when compared to states that are farther away. These are states which are better than its neighbours but later on gets worse.
- Plateau – A flat area of the search space, in which all neighbours have the same value. As all states have same value it is not possible to find the best direction.
- Ridge – It is a special kind of local maximum. It is a search area which is better than its neighbours but later on the values after selecting it gets worse.

To overcome these problems in hill climbing we adopt one of the following combination of methods:

- Backtracking for local maximum
- A big jump is the solution to escape from the plateau.
- Trying different paths at the same time is the solution for ridges.

Best First search

- Best first search is a way of combing the advantages of both depth first and breadth first search into a single method.

- Best first search is implemented using a **priority queue**.

Best first search procedure is very similar to hill climbing except for:

- In Hill climbing, one move is selected and all the other moves are rejected, never to be considered again.
- In Best first search, one move is selected, but the others are kept around so that they can be revisited later if the selected path becomes less promising.

A *

- A* is used on an OR graph.
- For finite state spaces A* will always terminate
- If A* is admissible i.e. if there is a path from start state to goal state then A* terminates by finding an optimal path.
- If there is a good underestimate then heuristic search A* is best.
- Cost function work as f(n) = g(n) + h(n), f(n) is cost of state, g(n) is cost from start node to node n, h(n) is heuristic function which is the estimated cost of path from node n to the goal.
- In A* f() is called fitness number.

AO *

- AO* is used on an AND OR graph. A* is not proper for AND/OR because for AND graphs all branches of it must be scanned to arrive at a solution.
- If there is a path from start state to goal state then AO* terminates but the path may or may not be an optimal path.
- Cost function work as f(n) = h(n).
- In AO * f() is called as futility, which is the cost incurred to solve a problem. If the estimated cost of a solution becomes greater than futility then we abandon the search.

Constraint satisfaction problem

- In real world environment there exists a lot of constraints. We need to find solutions without violating the constraints.

Problems that come under constraint satisfaction are:

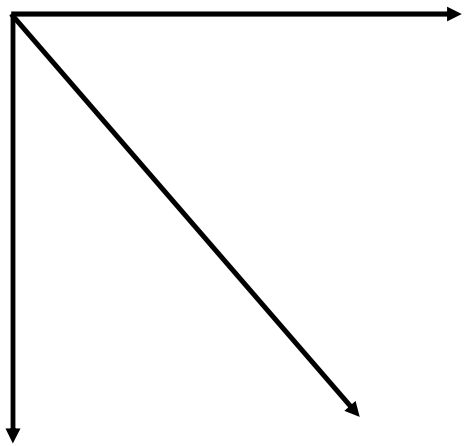
- Cryptarithmetic
- 8 queens
- Map colouring

Cryptarithmetic Problem

- Constraints
  - All alphabets must have different numeric values.
  - Numbers must not begin with zero
  - As addition operation is involved. Rules of addition are to be used.
- Goal state
  - Each letter is assigned a unique numeric value from 0 to 9.
- Heuristic
  - If there is a letter that has only two possible values and another having 6 possible values then we can make correct guess for the first one.
  - Assign values to letter that occur more frequently first
- Solution - It proceeds in cycles. At each cycle two things are done:
  - Constraints are propagated by using rules of arithmetic
  - A value is guessed for some letter whose value is not yet determined

8 Queen Problem

How to place 8 queen in a chess board so that none attacks the other.
A queen can move horizontally, vertically and diagonally.

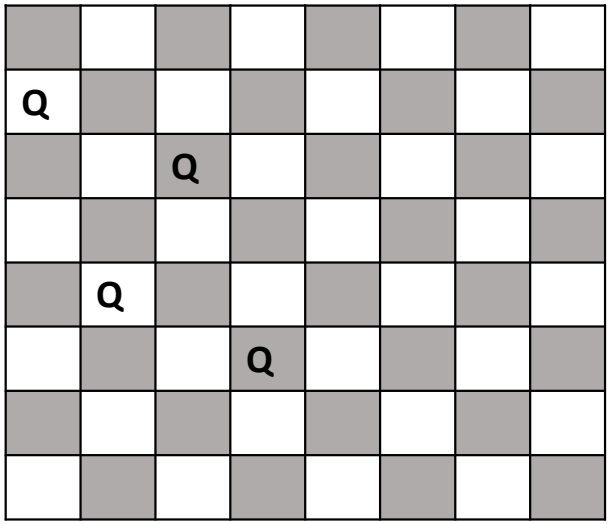


8 Queen Problem

There are three formulation:

Formulation I

- State space – It is any arrangement of 0 to 8 queens on board.
- Operators – Add a queen to any square where it does not attack the other, here we are not going row wise or column wise it is just at random we are placing the queen on board.
- Initial state – No queens on the board
- Goal state – 8 queens on the board, none attacked.



8 Queen Problem

Formulation II

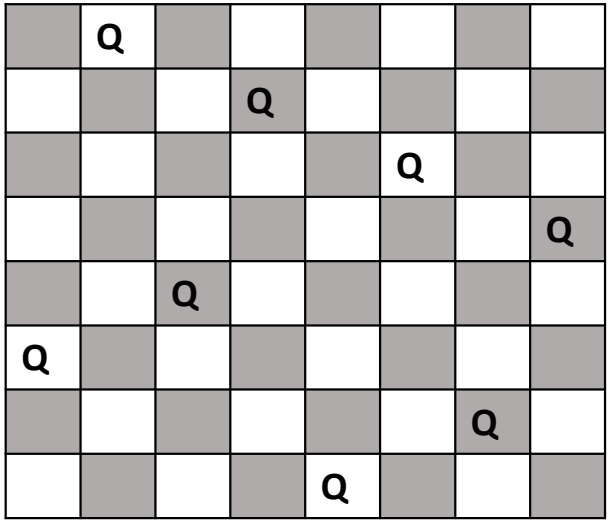
- State space – It is any arrangement of 0 to 8 queens with none attacked.
- Operators – Place the queen in the leftmost empty column.

➢ Place the first queen on the first column

➢ Place the second queen in the second column in a way that it does not attack the first queen

➢ Place the third queen in the square of third column so that it is not attacked by the first two and so on.

In this formulation we go from left to right and in a sequence we place queen column wise were none attack the other.

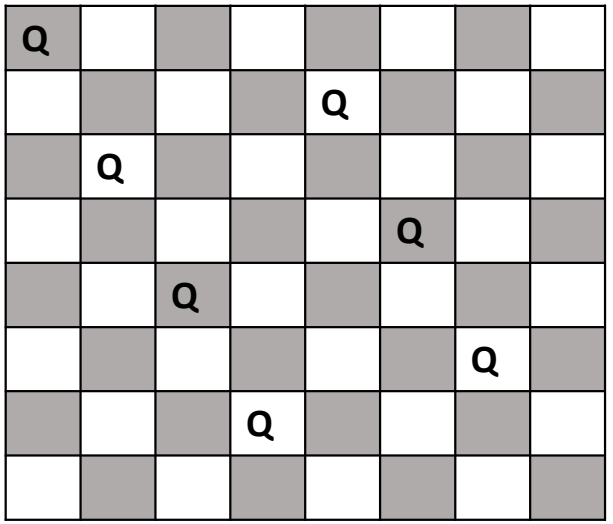


8 Queen Problem

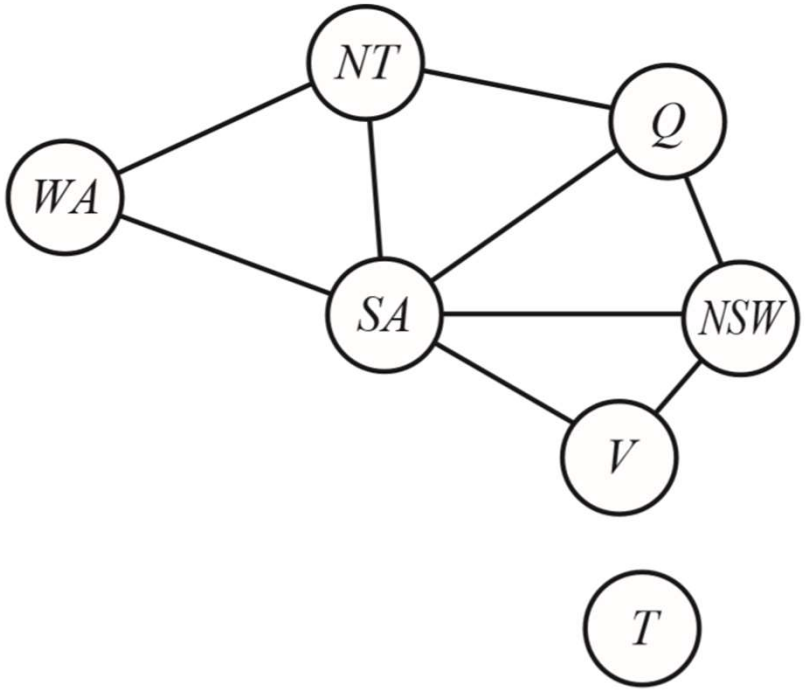
Formulation III

- State space – It is any arrangement of 8 queens one in each column.
- Operators – Move an attacked queen to another square in the same column.

In this formulation without looking at any of the constraint we place 8 queens on the board one in each column so that they attack each other and then the operators move an attacked queen to another square in the same column.



Map Colouring



The goal is to assign colours to each region so that no neighbouring regions have the same colour. The map-colouring problem is represented as a constraint graph.

If we choose {SA = blue} in the problem, then the five neighboring variables cannot take on the value blue.

Map Colouring

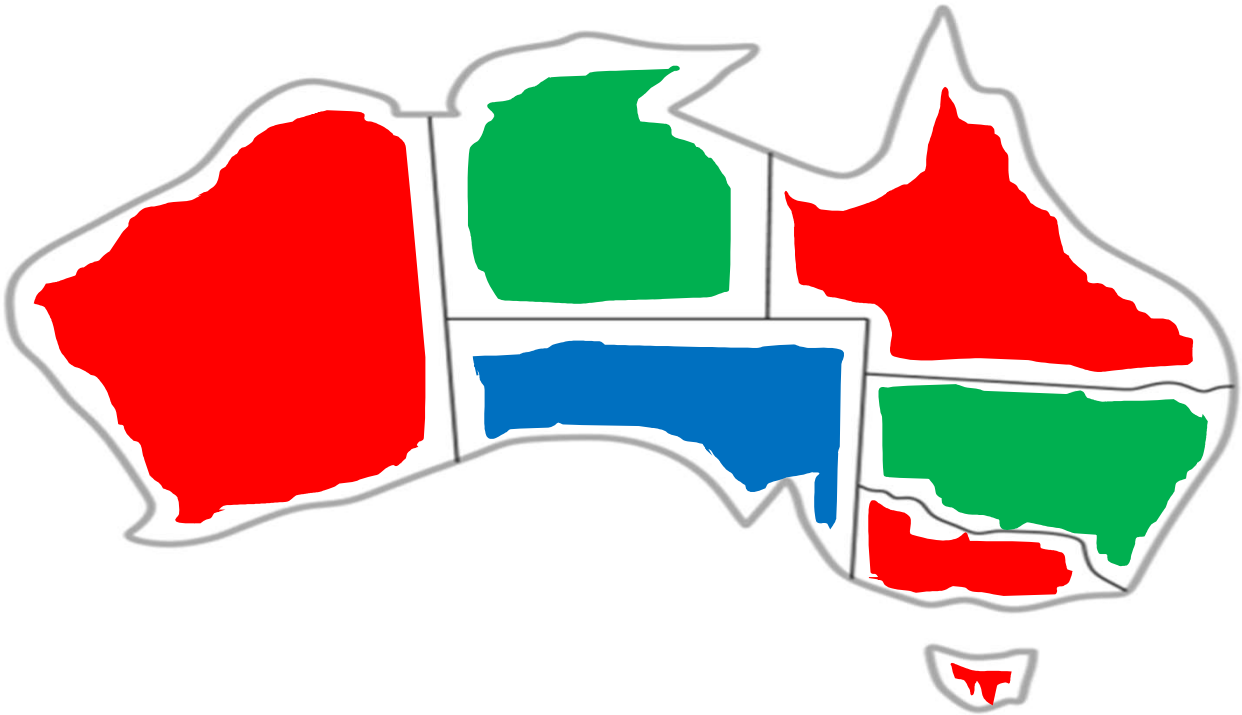
- In map colouring the task is to colour each region either red, green, or blue in such a way that no neighbouring regions have the same colour.
- The constraints require neighboring regions to have distinct colors. Since there are nine places where regions border, there are nine constraints:

$Constraint = \{SA \neq WA,\ SA \neq NT,\ SA \neq Q,\ SA \neq NSW,\ SA \neq V,\ WA \neq NT,\ NT \neq Q,\ Q \neq NSW,\ NSW \neq V\}$

- There are many possible solutions to this problem, such as

{SA = blue, WA = red, NT = green, Q = red, NSW = green, V =red, T = red }.

Map Colouring



Solution: SA = blue, WA = red, NT = green, Q = red, NSW = green, V =red, T = red

Game Playing

- In this we have multiagent environment.

- Basic characteristics of this strategy is look ahead in nature that is we explore tree for two or more levels downward and choose the optimal one. Basic method available for game playing is MINIMAX SEARCH.

Minimax Search

- It is a simple look ahead strategy. In look ahead strategy we have states generated up to some levels.

- We have game with two players whom we call MAX and MIN. Max moves first and then they take turns moving until the game is over. Max choose the maximum and aim of Min is to choose the minimum.

- Minimax search is depth first, so we consider nodes along a single path in a tree.

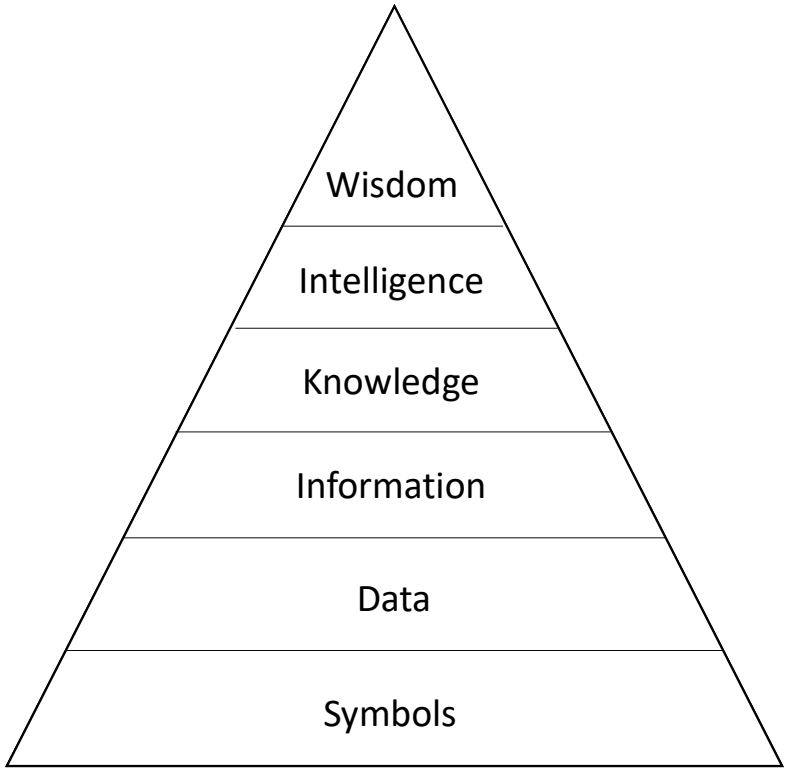
Knowledge Representation

UNIT III

Knowledge Pyramid

Knowledge is the underlying force behind every intelligent system.

- At the base of knowledge pyramid are symbols which forms means of representation.
- Data can be defined as a collection of mere symbols.
- Information is obtained after processing of data.
- Knowledge is organized information i.e. it is a piece of information that helps in decision making.
- Intelligence is the ability to draw useful inferences from the available knowledge.
- Wisdom is the maturity of the mind that directs its intelligence to achieve desirable goals.



Knowledge Pyramid

Knowledge Representation

- Knowledge is the collection of facts, inference rules etc. Knowledge requires the use of data and information. It combines relationships, dependencies with data and information.

- Knowledge is different from data. Data is the collection of raw materials whereas knowledge is the collection of some well specifies inference rules and facts.

- Knowledge is useless unless there is also an inference procedure that can exploit it.

Knowledge Representation

To solve a complex problem one needs:

- Both a large amount of knowledge
- Mechanism for manipulating that knowledge

There are two entities in knowledge representation:

- Facts – These are the things that we want to represent.
- Representations of facts – These are the things that we will be actually able to manipulate.

These two entities are structures at two levels:

- Knowledge level – At which facts are described.
- Symbol level – Object at the knowledge level are defined in terms of symbols that can be manipulated by programs.

Mappings between facts and representations

- Two way mapping exist between facts and representations.
- Forward representation mappings maps from facts to representations.
- Backward representation mapping goes from representations to facts.

Approaches to Knowledge Representation

A good system for the representation of knowledge should possess the following four properties:

- Representations Adequacy – ability to represent all knowledge kinds needed in that domain.
- Inferential Adequacy – ability to manipulate the representational structures in such a way as to derive new structure.
- Inferential Efficiency – ability to incorporate additional information into the knowledge structure.
- Acquisitional Efficiency – ability to acquire new information easily. The simplest case involves direct insertion of new knowledge by a person into the database.

Types/Techniques of knowledge Representation

- Simple relational knowledge
- Inheritable knowledge
- Inferential knowledge
- Procedural knowledge

Issues in Knowledge representation

- **Important Attributes** – Any attribute of objects so much basic that they occur in almost every problem domain?
  - Two attributes instance and isa are important attributes because they support property inheritance. They represent class membership and class inclusion.

- **Relationships between attributes** – Any important relationship that exists among object attributes?
  - The attributes that we use to describe objects are themselves entities that we represent. We need to know what properties they have which is independent of the knowledge they encode? There are four such properties:
  - Inverses
  - Existence in an isa hierarchy
  - Techniques for reasoning about values
  - Single valued attributes

Issues in Knowledge representation

- **Granularity** – At what level of detail should the knowledge be represented and what are the primitives?
  - Choosing the Granularity of Representation Primitives are fundamental concepts. The question arises – 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.
  - ➢ We have fact: John spotted Sue.
  - ➢ This could be represented as: spotted(agent(John), object(Sue))
  - ➢ Such a representation would make it easy to answer question as: Who spotted Sue?
  - ➢ But if we want to know: Did John see Sue?
  - ➢ The obvious answer is "yes" but given only the fact, we cannot discover the answer. So we need to add another fact as: $spotted(x, y) \rightarrow saw(x, y)$
  - ➢ Now we could infer the answer to the question.

Issues in Knowledge representation

- **Set of objects** – certain properties of objects that are true as member of a set but not as individual
  - Example – assert large(elephant);
  - In this there is distinction between:
  - Whether we are asserting some property of the set itself, means, *the set of elephants is large*, or
  - Asserting some property that holds for individual elements of the set, means, *anything that is an elephant is large*.

Issues in Knowledge representation

- **Finding right structure** – Given a large amount of knowledge stored, how can relevant parts can be accessed?
  - Access to right structure for describing a particular situation. It requires, selecting an initial structure and then revising the choice. While doing so it is necessary to solve following problems:
  - How to perform an initial selection of the most appropriate structure.
  - How to fill in appropriate details from the current situations.
  - How to find a better structure if the one chosen initially turns out not to be appropriate.
  - What to do if none of the available structures is appropriate.
  - When to create and remember a new structure.