Unit I Introduction

1. Artificial Intelligence:
   1.1 Introduction
   1.2 Typical Applications
2. State Space Search:
   2.1 Depth Bounded DFS
   2.2 Depth First Iterative Deepening
3. Heuristic Search:
   3.1 Heuristic Functions
   3.2 Best First Search
   3.3 Hill Climbing
   3.4 Variable Neighbourhood Descent
   3.5 Beam Search
   3.6 Tabu Search
4. Optimal Search:
   4.1 A* algorithm
   4.2 Iterative Deepening A*
   4.3 Recursive Best First Search
   4.4 Pruning the CLOSED and OPEN Lists

1 Artificial Intelligence:

- Artificial intelligence (AI) is an area of computer science that emphasizes the creation of intelligent machines that work and reacts like humans. Some of the activities computers with artificial intelligence are designed.
- Since the invention of computers or machines, their capability to perform various tasks went on growing exponentially.
- Humans have developed the power of computer systems in terms of their diverse working domains, their increasing speed, and reducing size with respect to time.
- A branch of Computer Science named Artificial Intelligence pursues creating the computers or machines as intelligent as human beings.
- The ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings.
- The term is frequently applied to the project of developing systems endowed with the intellectual processes characteristic of humans, such as the ability to reason.

1.1 Introduction

- Artificial Intelligence is an approach to make a computer, a robot, or a product to think how smart human think.
- AI is a study of how human brain think, learn, decide and work, when it tries to solve problems. And finally this study outputs intelligent software systems.
- The aim of AI is to improve computer functions which are related to human knowledge, for example, reasoning, learning, and problem-solving.

The intelligence is intangible and composed of

- Reasoning
- Learning
- Problem Solving
- Perception
- Linguistic Intelligence

- The objectives of AI research are reasoning, knowledge representation, planning, learning, natural language processing, realization, and ability to move and manipulate objects. There are long-term goals in the general intelligence sector.

- Artificial Intelligence also called (AI), which is an imitation of human intelligence system processes by a computer or an android.

- It includes processes to acquire information, validate assigned rules, and debug itself by the intelligence software and execute actions by the hardware installed with it.
- Let's simplify the definition of AI, a machine, robot or android that has an intelligence system like a human brain which can Sense, Reason, Act & Adapt according to the operational instructions. It works based on the data stored and configured with real-time instructions.

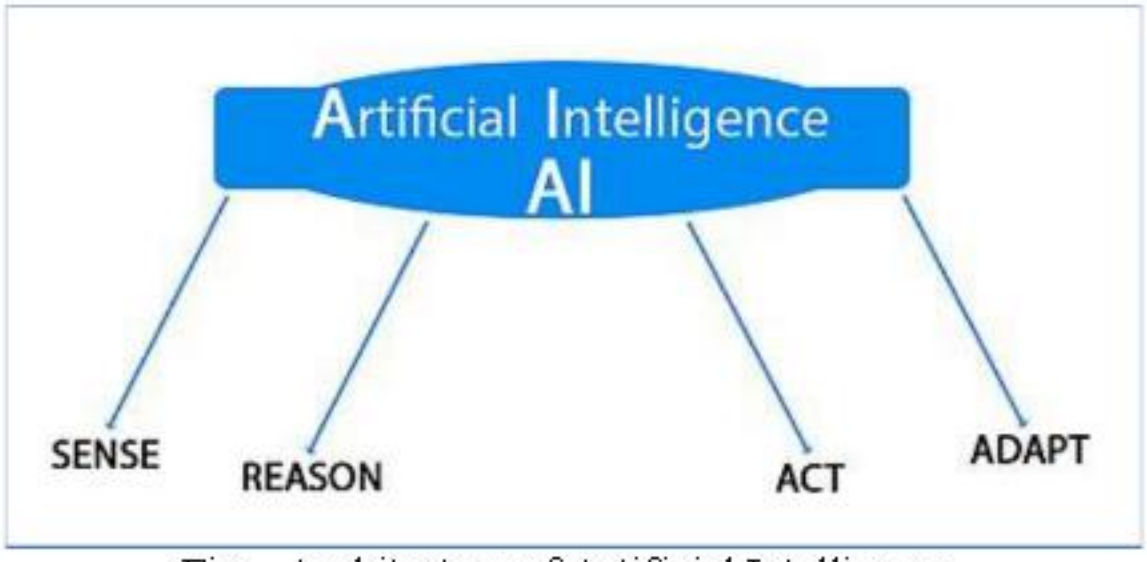

Fig – Architecture of Artificial Intelligence

1.2 Typical Applications

- Artificial Intelligence, defined as intelligence exhibited by machines, has many applications such as more specifically. The form of AI where programs are developed to perform specific tasks.

- That is being utilized for a wide range of activities including medical diagnosis, Electronic trading platforms, Robot control and remote sensing.

- AI has been used to develop and advance numerous fields and industries, including finance, healthcare, education, transportation, etc.

Typical Applications of AI

- **Gaming** – AI plays crucial role in strategic games such as chess, poker, tic-tac-toe, etc., where machine can think of large number of possible positions based on heuristic knowledge.

- **Natural Language Processing** – It is possible to interact with the computer that understands natural language spoken by humans.

- **Expert Systems** – There are some applications which integrate machine, software, and special information to impart reasoning and advising. They provide explanation and advice to the users.

- **Vision Systems** – These systems understand, interpret, and comprehend visual input on the computer. For example,

  - A spying aeroplane takes photographs, which are used to figure out spatial information or map of the areas.

  - Doctors use clinical expert system to diagnose the patient.

  - Police use computer software that can recognize the face of criminal with the stored portrait made by forensic artist.

- **Speech Recognition** – Some intelligent systems are capable of hearing and comprehending the language in terms of sentences and their meanings while a human talks to it. It can handle different accents, slang words, noise in the background, change in human's noise due to cold, etc.

- **Handwriting Recognition** – the handwriting recognition software reads the text written on paper by a pen or on screen by a stylus. It can recognize the shapes of the letters and convert it into editable text.

- **Intelligent Robots** – Robots are able to perform the tasks given by a human. They have sensors to detect physical data from the real world such as light, heat, temperature, movement, sound, bump, and pressure. They have efficient processors, multiple sensors and huge memory, to exhibit intelligence. In addition, they are capable of learning from their mistakes and they can adapt to the new environment.

2 State Space Search:

- State space search is a process used in the field of computer science, including artificial intelligence (AI), in which successive configurations or states.

- State space search often differs from traditional computer science search methods because the state space is implicit: the typical state space graph is much too large to generate and store in memory.

- Instead, nodes are generated as they are explored, and typically discarded thereafter. A solution to a combinatorial search instance may consist of the goal state itself, or of a path from some initial state to the goal state.

Examples of state-space search

➢ **Uninformed Search**

The following are uninformed state-space search methods, meaning that they do not know information about the goal's location.

- Traditional depth-first search
- Breadth-first search
- Iterative deepening
- Lowest-cost-first search

➢ **Heuristic Search**

Some algorithms take into account information about the goal node's location in the form of a heuristic function. Poole and Mackworth cite the following examples as informed search algorithms:

- Heuristic depth-first search
- Greedy best-first search
- A* search

2.1 Depth Bounded DFS

- Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node.

- A depth-first search starting at A, assuming that the left edges in the shown graph are chosen before right edges, and assuming the search remembers previously visited nodes and will not repeat them (since this is a small graph), will visit the nodes in the following order:

- A, B, D, F, E, C, G. The edges traversed in this search form a a structure with important applications in graph theory. Performing the same search without remembering previously visited nodes results in visiting nodes in the order A,

B, D, F, E, A, B, D, F, E, etc. forever, caught in the A, B, D, F, E cycle and never reaching C or G.

Example

- For the following graph:

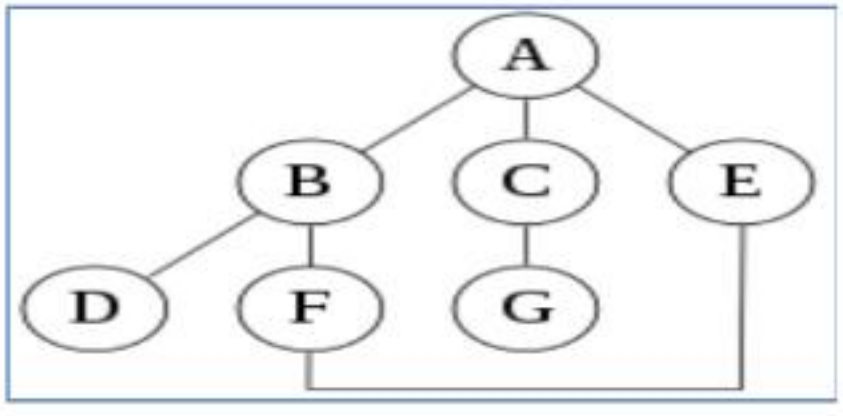

Fig - Depth Bounded DFS

- In above graph A, B, D, F, E, C, and G. The edges traversed in this search form a structure with important applications performing the same search without remembering previously visited nodes results in visiting nodes.
- In the order A,B, D, F, E, A, B, D, F, E, etc. forever, caught in the A, B, D, F, E cycle an never reaching C or G as shown in above example.

2.2 Depth First Iterative Deepening

- Iterative deepening search or more specifically iterative deepening depth-first search (IDS or IDDFS) is a state space/graph search strategy in which a depth-limited version of depth-first search is run repeatedly with increasing depth limits until the goal is found.
- IDDFS is optimal like breadth-first search, but uses much less memory; at each iteration, it visits the nodes in the search tree in the same order as depth-first search, but the cumulative order in which nodes are first visited is effectively breadth-first.
- A second advantage is the responsiveness of the algorithm. Because early iterations use small values for they execute extremely quickly. This allows the algorithm to supply early indications of the result almost immediately, followed by refinements as increases.

- When used in an interactive setting, such as in a chess-playing program, this facility allows the program to play at any time with the current best move found in the search it has completed so far.

- This can be phrased as each depth of the search producing a better approximation of the solution, though the work done at each step is recursive. This is not possible with a traditional depth-first search, which does not produce intermediate results.

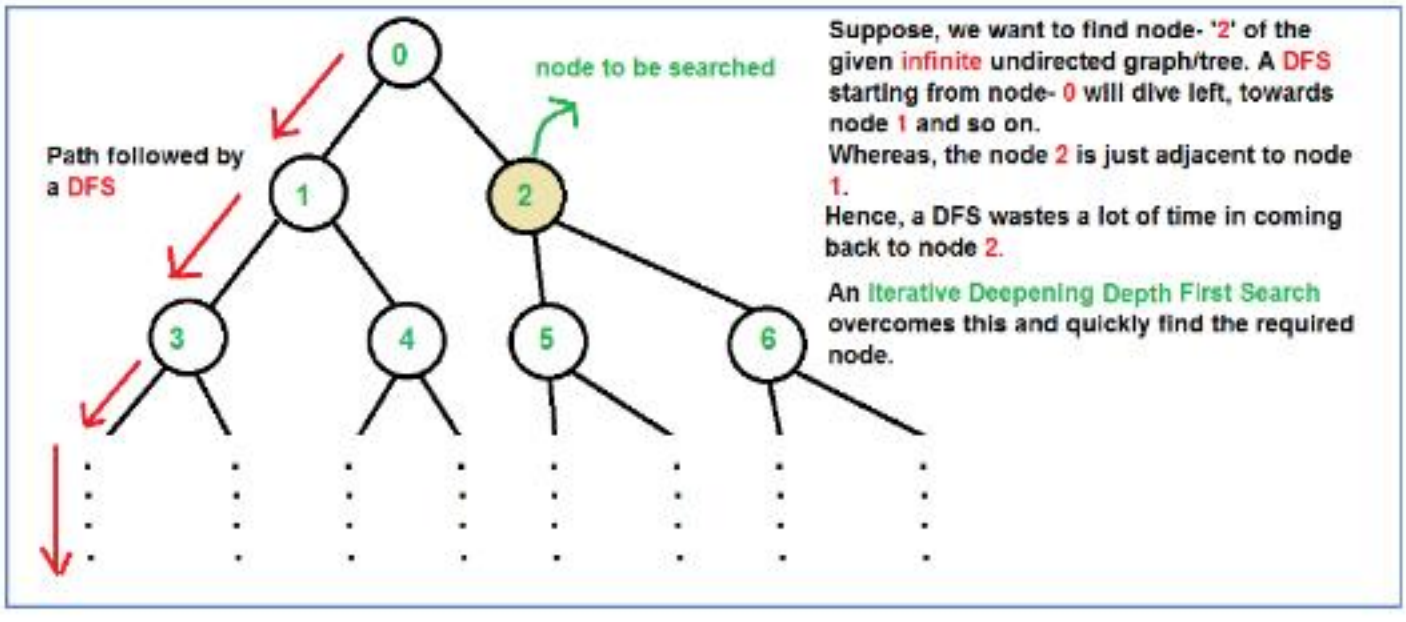

Fig- Depth First Iterative Deepening

- DDFS combines depth-first search's space-efficiency and breadth-first search's completeness (when the branching factor is finite). If a solution exists, it will find a solution path with the fewest number of arcs.

- Since iterative deepening visits states multiple times, it may seem wasteful, but it turns out to be not so costly, since in a tree most of the nodes are in the bottom level, so it does not matter much if the upper levels are visited multiple times.

- The main advantage of IDDFS in game tree searching is that the earlier searches tend to improve the commonly used heuristics, such as the killer heuristic and alpha-beta pruning, so that a more accurate estimate of the score of various nodes at the final depth search can occur, and the search completes more quickly since it is done in a better order. For example, alpha-beta pruning is most efficient if it searches the best moves first.

- A second advantage is the responsiveness of the algorithm. Because early iterations use small values for they execute extremely quickly. This allows the algorithm to supply early indications of the result almost immediately, followed by refinements as increases.

- When used in an interactive setting, such as in a chess-playing program, this facility allows the program to play at any time with the current best move found in the search it has completed so far.
- This can be phrased as each depth of the search corecursively producing a better approximation of the solution, though the work done at each step is recursive

3 Heuristic Search:

- In artificial intelligence, and mathematical optimization, a heuristic is a technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution.
- This is achieved by trading optimality, completeness, accuracy, or precision for speed. In a way, it can be considered a shortcut. A heuristic function, also called simply a heuristic, is a function that ranks alternatives in search.
- The objective of a heuristic is to produce a solution in a reasonable time frame that is good enough for solving the problem at hand. This solution may not be the best of all the solutions to this problem, or it may simply approximate the exact solution. But it is still valuable because finding it does not require a prohibitively long time.
- Heuristics may produce results by themselves, or they may be used in conjunction with optimization algorithms to improve their efficiency (e.g., they may be used to generate good seed values).
- Results about NP-hardness in theoretical computer science make heuristics the only viable option for a variety of complex optimization problems that need to be routinely solved in real-world applications.
- Heuristics underlie the whole field of Artificial Intelligence and the computer simulation of thinking, as they may be used in situations where there are no known algorithms.

3.1 Heuristic Functions

- A heuristic function, also called simply a heuristic, is a function that ranks alternatives in search algorithms.
- The heuristic function is a way to inform the search about the direction to a goal. It provides an informed way to guess which neighbor of a node will lead

to a goal.

- There is nothing magical about a heuristic function. It must use only information that can be readily obtained about a node.

- A heuristic function that ranks alternatives in search algorithms at each branching step based on available information to decide which branch to following example.

- The employ heuristics include using a rule of thumb, an educated guess, an intuitive judgment, a guesstimate, profiling, or common sense. There is nothing magical about a heuristic function. It must use only information that can be readily obtained about a node

Types of heuristic functions

- Availability,
- representativeness and
- anchoring and adjustment

3.2 Best First Search

- Best-first search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule.
- "Best-first search" to refer specifically to a search with a heuristic that attempts to predict how close the end of a path is to a solution, so that paths which are judged to be closer to a solution are extended first.
- This specific type of search is called greedy best-first search or pure heuristic search.
- Efficient selection of the current best candidate for extension is typically implemented using a priority queue.
- The A* search algorithm is an example of a best-first search algorithm, as is B*. Best-first algorithms are often used for path finding in combinatorial search. Neither A* nor B* is a greedy best-first search, as they incorporate the distance from the start in addition to estimated distances to the goal.

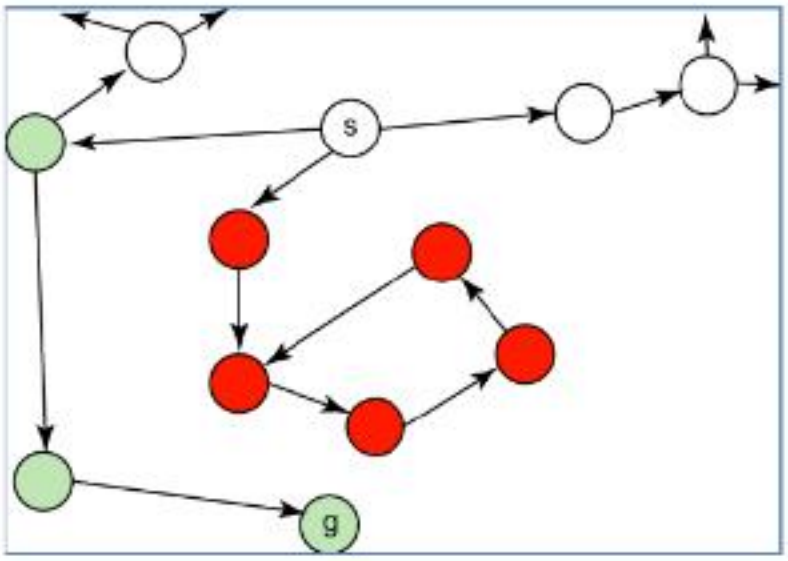

Fig- Best-first search

- Another way to use a heuristic function is to always select a path on the frontier with the lowest heuristic value. This is called **best-first search**. It usually does not work very well; it can follow paths that look promising because they are close to the goal, but the costs of the paths may keep increasing.

3.3 Hill Climbing

- Hill climbing is a mathematical optimization technique which belongs to the family of local search. It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by making an incremental change to the solution.
- If the change produces a better solution, another incremental change is made to the new solution, and so on until no further improvements can be found.
- For example, hill climbing can be applied to the travelling salesman problem. It is easy to find an initial solution that visits all the cities but will likely be very poor compared to the optimal solution.
- It is an iterative algorithm that starts with an arbitrary solution to a problem and attempts to find a better solution by changing a single element of the solution incrementally.
- If the change produces a better solution, an incremental change is taken as a new solution. This process is repeated until there are no further improvements. Function Hill-Climbing, returns a state that is a local maximum.

Applications

- Hill climbing can be applied to any problem where the current state allows for an accurate evaluation function. For example, the travelling salesman problem,

the eight-queens problem, circuit design, and a variety of other real-world problems. Hill Climbing has been used in inductive learning models.

3.4 Variable Neighbourhood Descent

- Two variants of a variable neighbourhood descent algorithm are proposed for solving the MPSP with metal uncertainty. The proposed methods are tested and compared on actual large-scale instances, and very good solutions.

- Specifically, we introduce a metaheuristic method based on a variable neighbourhood descent (VND) procedure. To generate the initial solution to be improved by this procedure, we consider two different alternatives. Both are based on a decomposition approach separating the problem into a series of sub-problems, each associated with one period.

- Recall: Local minima are relative to neighbourhood relation. I key idea: To escape from local minimum of given neighbourhood relation, switch to di↵erent neighbhourhood relation. I use k neighbourhood relations N1,..., Nk , (typically) ordered according to increasing neighbourhood size.

- Smallest neighbourhood that facilitates improving steps. I upon termination, candidate solution am locally optimal with respect to. all neighbourhoods. Determine initial candidate solution s i := 1 Repeat: | choose a most improving neighbour s0 of s in Ni | If g(s0) < g(s): | s := s0 | i := 1 | Else: | i := i + 1 Until i > k

- Variable neighbourhood search (VNS), proposed by Mladenović, Hansen, 1997 is a met heuristic method for solving a set of combinatorial optimization and global optimization problems.

- It explores distant neighbourhoods of the current incumbent solution, and moves from there to a new one if and only if an improvement was made. The local search method is applied repeatedly to get from solutions in the neighbourhood to local optima.

- VNS was designed for approximating solutions of discrete and continuous optimization problems and according to these, it is aimed for solving linear program problems, integer program problems, mixed integer program problems, nonlinear program problems, etc.

Applications

- Industrial applications
- Design problems in communication
- Location problems
- Data mining
- Graph problems
- Knapsack and packing problems
- Mixed integer problems
- Time tabling
- Scheduling
- Vehicle routing problems
- Arc routing and waste collection
- Fleet sheet problems
- Extended vehicle routing problems
- Problems in biosciences and chemistry
- Continuous optimization
- Other optimization problems
- Discovery science

3.5 Beam Search

- Beam search uses breadth-first search to build its search tree. At each level of the tree, it generates all successors of the states at the current level, sorting them in increasing order of heuristic cost.
- A beam search is most often used to maintain tractability in large systems with insufficient amount of memory to store the entire search tree. For example, it is used in many machine translation systems.
- However, it only stores a predetermined number of best states at each level (called the beam width). Only those states are expanded next. The greater the beam width, the fewer states are pruned. With an infinite beam width, no states are pruned and beam search is identical to breadth-first search.

- The beam width bounds the memory required to perform the search. Since a goal state could potentially be pruned, beam search sacrifices completeness. Beam search is not optimal (that is, there is no guarantee that it will find the best solution).
- In general, beam search returns the first solution found. Beam search for machine translation is a different case: once reaching the configured maximum search depth (i.e. translation length), the algorithm will evaluate the solutions found during search at various depths and return the best one (the one with the highest probability).
- The beam width can either be fixed or variable. One approach that uses a variable beam width starts with the width at a minimum. If no solution is found, the beam is widened and the procedure is repeated.
- To select the best translation, each part is processed, and many different ways of translating the words appear. The top best translations according to their sentence structures are kept and the rest are discarded.
- The translator then evaluates the translations according to a given criterion, choosing the translation which best keeps the goals.

3.6 Tabu Search

- Tabu search, created by Fred W. Glover in 1986 and formalized in 1989, is a search method employing local search methods used for mathematical optimization.
- Local (neighborhood) searches take a potential solution to a problem and check its immediate neighbours (that is, solutions that are similar except for very few minor details) in the hope of finding an improved solution. Local search methods have a tendency to become stuck in suboptimal regions or on plateaus where many solutions are equally fit.
- Tabu search enhances the performance of local search by relaxing its basic rule. First, at each step worsening moves can be accepted if no improving move is available like when the search is stuck at a strict local minimum. In addition, prohibitions (henceforth the term tabu) are introduced to discourage the search from coming back to previously-visited solutions.
- Tabu was first conceived of by Fred Glover while observing his graduate classmates trying to emulate problem solving on computers in the early 1960s (Glover 1998). During this time he noticed that his colleagues did not use formal logic for problem solving themselves but would instead whittle down the

problem space by avoiding previously tried steps if they proved unpromising (Glover 1998).

- The name Tabu comes from the word taboo which adequately describes the above phenomena. Tabu is a heuristic search algorithm used to solve combinatorial optimization problems, where a set of discrete feasible solutions is the problem space and the goal is to find the best possible solution (Glover 1989).
- In other words, it is a metaheuristic that aids local search heuristics to reach optimal performance while searching for a solution. One of Tabu's uniqueness's is attributed to its flexible memory system, as opposed to other search methods that have a fixed or no memory system.

Types of memory

- The memory structures used in tabu search can roughly be divided into three categories:
- Short-term: The list of solutions recently considered. If a potential solution appears on the tabu list, it cannot be revisited until it reaches an expiration point.
- Intermediate-term: Intensification rules intended to bias the search towards promising areas of the search space.
- Long-term: Diversification rules that drive the search into new regions

4 Optimal Search:

- An optimal search tree (Optimal ST), sometimes called a weight-balanced binary tree, is a binary search tree. Optimal search provides the smallest possible search time (or expected search time) for a given sequence of access.
- Optimal BSTs are generally divided into two types: static and dynamic. In the static optimality problem, the tree cannot be modified after it has been constructed.
- In this case, there exists some particular layout of the nodes of the tree which provides the smallest expected search time for the given access probabilities. Various algorithms exist to construct or approximate the statically optimal tree given the information on the access probabilities of the elements.
- In the dynamic optimality problem, the tree can be modified at any time, typically by permitting tree rotations.

- The tree is considered to have a cursor starting at the root which it can move or use to perform modifications. In this case, there exists some minimal-cost sequence of these operations which causes the cursor to visit every node in the target access sequence in order.
- The splay tree is conjectured to have a constant competitive ratio compared to the dynamically optimal tree in all cases, though this has not yet been proven.
- Optimal binary search tree (Optimal BST), sometimes called a weight-balanced binary tree, is a binary search tree which provides the smallest possible search time (or expected search time) for a given sequence of accesses. Optimal BSTs are generally divided into two types: static and dynamic.
- In the static optimality problem, the tree cannot be modified after it has been constructed. In this case, there exists some particular layout of the nodes of the tree which provides the smallest expected search time for the given access probabilities. Various algorithms exist to construct or approximate the statically optimal tree given the information on the access probabilities of the elements.

4.1 A* algorithm

- A* was created as part of the Shakey's project, which had the aim of building a mobile robot that could plan its own actions. Nils Nilsson originally proposed using the Graph Traverse algorithm for Shakey's path planning. Graph Traverse is guided by a heuristic function the estimated distance from node.
- The goal node, it entirely ignores the distance from the start node to Bertram Raphael suggested using the sum; Peter Hart invented the concepts we now call admissibility and consistency of heuristic functions. A* was originally designed for finding least-cost paths when the cost of a path is the sum of its edge costs, but it has been shown that A* can be used to find optimal paths for any problem satisfying the conditions of a cost algebra.
- A* is an informed search algorithm, or a best-first search, meaning that it is formulated in terms of weighted graphs: starting from a specific starting node of a graph, it aims to find a path to the given goal node having the smallest cost.
- It does this by maintaining a tree of paths originating at the start node and extending those paths one edge at a time until its termination criterion is satisfied.
- At each iteration of its main loop, A* needs to determine which of its paths to extend. It does so based on the cost of the path and an estimate of the cost

required extending the path all the way to the goal. Specifically, A* selects the path that minimizes.

Applications

- A* is commonly used for the common path finding problem in applications such as video games, but was originally designed as a general graph traversal algorithm.
- It finds applications to diverse problems, including the problem of parsing using stochastic grammars in NLP.
- Other cases include an Informational search with online learning.

4.2 Iterative Deepening A*

- Iterative deepening A* (IDA*) is a graph traversal and path search algorithm that can find the shortest path between a designated start node and any member of a set of goal nodes in a weighted graph.
- It is a variant of iterative deepening depth-first search that borrows the idea to use a heuristic function to evaluate the remaining cost to get to the goal from the A* search algorithm. Since it is a depth-first search algorithm.
- Its memory usage is lower than in A*, but unlike ordinary iterative deepening search, it concentrates on exploring the most promising nodes and thus does not go to the same depth everywhere in the search tree. Unlike A*, IDA* does not utilize dynamic programming.
- Typical implementations of A* use a priority queue to perform the repeated selection of minimum (estimated) cost nodes to expand. This priority queue is known as the open set or fringe.
- At each step of the algorithm, the node with the lowest f(x) value is removed from the queue, the f and g values of its neighbours are updated accordingly, and these neighbours' are added to the queue.
- The algorithm continues until a goal node has a lower f value than any node in the queue. The f value of the goal is then the cost of the shortest path, since h at the goal is zero in an admissible heuristic.
- The algorithm described so far gives us only the length of the shortest path. To find the actual sequence of steps, the algorithm can be easily revised so that each node on the path keeps track of its predecessor.

- After this algorithm is run, the ending node will point to its predecessor, and so on, until some node's predecessor is the start node.

Example

An example of an A* algorithm in action where nodes are cities connected with roads and h(x) is the straight-line distance to target point:

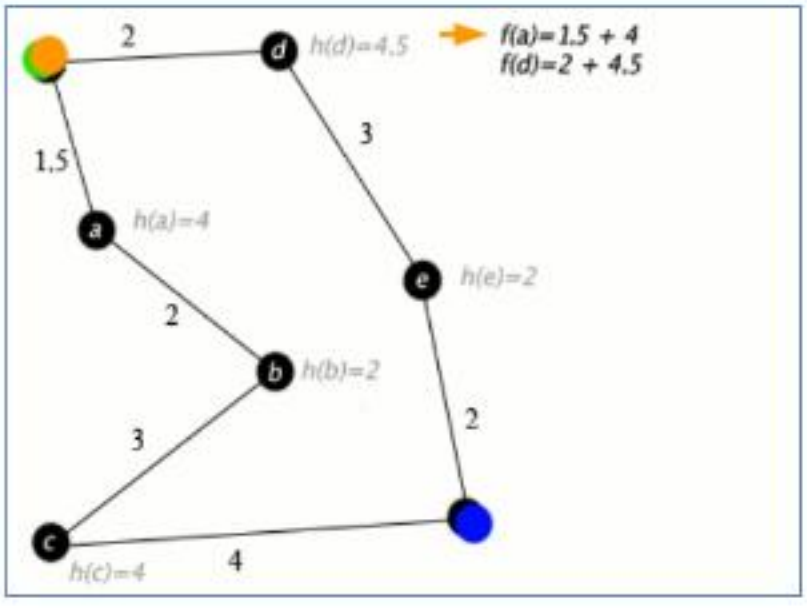

Fig - Example of an A* algorithm

Key: green: start; blue: goal; orange: visited

The A* algorithm also has real-world applications. In this example, edges are railroads and h(x) is the great-circle distance (the shortest possible distance on a sphere) to the target. The algorithm is searching for a path between Washington, D.C. and Los Angeles.

4.3 Recursive Breadth First Search

- RBFS is a best-first search that runs in space that is linear with respect to the maximum search depth, regardless of the cost function used. Even with an admissible cost function, RBFS generates fewer nodes than IDA*, and is generally superior to IDA*, except for a small increase in the cost per node generation.
- We present a novel, RBFS based algorithm which uses all available memory. This is achieved by keeping the generated nodes in memory and pruning some of them away when the memory runs out. This new RBFS algorithm is implemented using Anytime Algorithm and will be called as Anytime Recursive best first search.

- Thus the RBFS algorithm backtracks and updating the node with the least f-cost of its successors.
- public double RBFS(Node Root, Node Goal, double limit)

```
{
if(Root==Goal)
{goalSucc=true; return Root.fCost; }
else
{
Node [ ] successors=ExpandNode (Root);
successors.SortNodes(); /*sorts successors nodes by increasing g fCost*/
if(successors[firstNode].fCost>limit)
return successors[firstNode].fCost;
else
{
closeList.Insert(Root); foreach(Node s in successors)
{
if(s!=closeList[item]) openList.Insert(s);
 }
openList.Sort();
Node bestNode=openList.RemoveFirstNode();
Node alternativeNode=openList.RemoveFirstNode();
while(goalSucc==false)
{
 bestNode=RBFS(bestNode,Goal,Math.Min(limit, alternativeNode.fCost));
openList.Insert(bestNode); list.Sort( );
bestNode=openList.RemoveFirstNode();
alternativeNode=openList.RemoveFirstNode();
}
```

```
}
}
}
```

- Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key' [1]), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.
- It uses the opposite strategy as depth-first search, which instead explores the highest-depth nodes first before being forced to backtrack and expand shallower nodes.
- BFS and its application in finding connected components of graphs were invented in 1945 by Konrad Zuse, in his (rejected) Ph.D. thesis on the Plankalkül programming language, but this was not published until 1972.

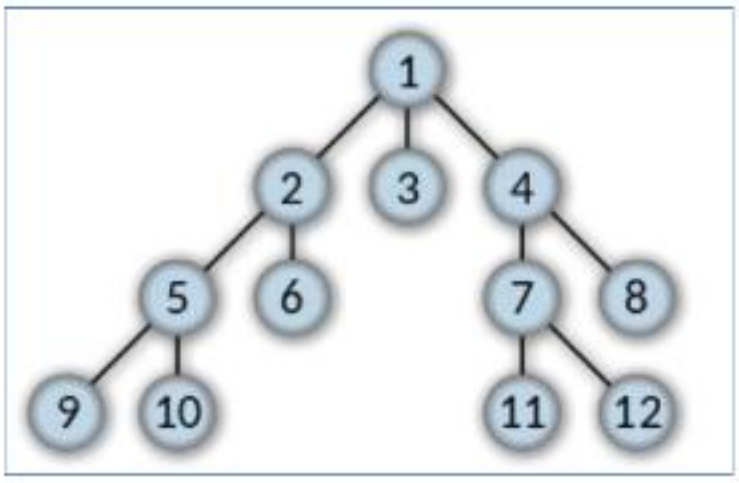

Fig- Breadth-first search (BFS).

- In the above fig. shows the parent attribute of each vertex is useful for accessing the nodes in a shortest path, for example by backtracking from the destination node up to the starting node, once the BFS has been run, and the predecessors nodes have been set.

  Breadth-first search produces a so-called breadth first tree.

  Applications

- Copying garbage collection, Cheney's algorithm
- Finding the shortest path between two nodes u and v, with path length measured by number of edges (an advantage over depth-first search [9]
- (Reverse) Cuthill–McKee mesh numbering

- Ford–Fulkerson method for computing the maximum flow in a flow network
- Serialization/Deserialization of a binary tree vs. serialization in sorted order, allows the tree to be re-constructed in an efficient manner.
- Construction of the failure functions of the Aho-Corasick pattern matcher.
- Testing bipartiteness of a graph.

4.4 Pruning the CLOSED and OPEN Lists

- In most search techniques that explore graphs, it is necessary to record two kinds of nodes: the nodes we have seen but not explored, and the nodes we have seen and explored. The OPEN list is the former, and the CLOSED list is the latter.

- Generally, search proceeds by examining each node on the OPEN list, performing some expansion operation that adds its children to the OPEN list, and moving the node to the CLOSED list. The open list is a collection of all generated nodes. This means that those are nodes that were neighbours of expanded nodes.

- As mentioned above, the open list is often implemented as a priority queue so the search can simply dequeue the nest best node.

- The set of all leaf nodes available for expansion at any given FRONTIER point is called the frontier. Many authors call it the open list, which is both geographically OPEN LISTS less evocative and inaccurate, as it need not be stored as a list at all. The closed list is a collection of all expanded nodes. This means that those are nodes that were already "searched".

- This prevents the search from visiting nodes again and again. A side note: in big domains, the closed list can't fit all nodes, so the closed list has to be implemented smartly. For example, it is possible to reduce the memory required using a Bloom Filter.

- The purpose of the closed list is to track which nodes have already been expanded. This ensures your algorithm terminates, not getting into an infinite loop by repeatedly expanding the same subset of nodes.

- Many authors call this the closed list see earlier comment on open lists. Newly generated nodes that match previously generated nodes ones in the explored set or the frontier can be discarded instead of being added to the frontier.

Reference:

1. Nilsson Nils J , "Artificial Intelligence: A new Synthesis, Morgan Kaufmann Publishers Inc. San Francisco, CA, ISBN: 978-1-55-860467-4
2. Patrick Henry Winston, "Artificial Intelligence", Addison-Wesley Publishing Company, ISBN: 0-201-53377-4.
3. Deepak Khemani, "A First Course in Artificial Intelligence", McGraw Hill Education (India), 2013, ISBN : 978-1-25-902998-1
4. Elaine Rich, Kevin Knight and Nair, "Artificial Intelligence", TMH, ISBN-978-0-07-008770-5
5. Stuart Russell and Peter Norvig, "Artificial Intelligence: A Modern Approach", Third edition, Pearson, 2003, ISBN :10: 0136042597

Unit II Problem Decomposition and Planning

1. Problem Decomposition:

1.1 Goal Trees

1.2 Rule Based Systems

1.3 Rule Based Expert Systems

2. Planning:

2.1 STRIPS

2.2 Forward and Backward State Space Planning

2.3 Goal Stack Planning

2.4 Plan Space Planning

2.5 A Unified Framework For Planning

3. Constraint Satisfaction:

3.1 N-Queens

3.2 Constraint Propagation

3.3 Scene Labelling

1. Problem Decomposition:

- The ability to decompose a complex problem into manageable subcomponents is a necessity to many intelligent problem-solving activities. A problem solver using a problem-decomposition or problem-reduction strategy would first decompose a given problem into sub problems, solve each sub problem and then combine the solutions to obtain a global solution to the original problem.

- In distributed problem solving, an agent could be assigned to solve each of the sub problems. The advantage of problem decomposition cannot be overstressed; it facilitates concurrent problem-solving and reduces search complexity.

- An important problem is how to compute problem decomposition automatically. So far, the problem has only been superficially considered. In the past, a number of nonlinear planning algorithms have been proposed, all decomposing a problem simply by splitting a compound goal into sub goals.

- Furthermore, no concurrent problem-solving is done; most planners solve all sub goals together.

Example
To begin with, we consider a simple example to illustrate the main points. Consider the following example where two boxes can be moved around three rooms. Figure 1 (a) depicts such a domain. Suppose that from room R2, the only way to transport any box through to room R3 is to use a cart to push the box together with one other box.

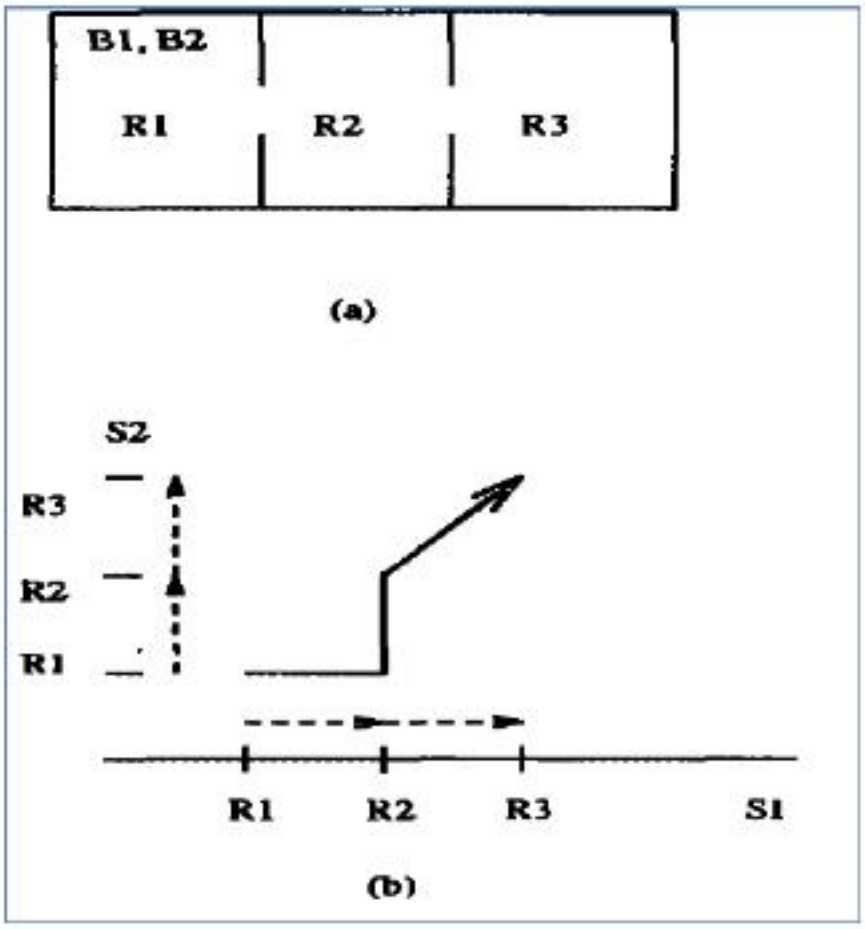

Fig - Example of Problem Decomposition

- In this domain, a typical conjunctive goal is to rearrange the boxes in different rooms. As an example, a goal state in which both boxes B1 and B2 are in room R3 can be described as: G1 ^ G2, where Gl=In room(B1,R3) and G2=In room(B2,R3). A conjunctive goal planner solves this problem by planning the two sub goals together.
- During the achievement of any one sub goal, a check must be made in the entire plan to see if any other goal is violated. In contrast, a planner based on a problem decomposition method separates the planning process for the movement of the two boxes.
- In each decomposed sub domain, it forms a solution plan for each box individually, and then combines the two plans to form a single global plan. In this example, this separation might correspond to solving each sub goal G1 or G2 concurrently.

1.1 Goal Trees

- A Goal Trees, sometimes still referred to as Intermediate Objective Map or IO Map, is primarily a Logical Thinking Process tool, itself linked to the Theory of Constraints. The top of the tree deals with strategic planning while going down to its bottom links strategy to operations.
- A goal tree is a diagram used to define the criteria for evaluating alternative solutions to a problem.

- It is drawn from the problems formulation. That is, the statement gathering an actor´s objective and the "unwanted downside" which avoids to fulfil the objective.
- On the basis of the objective stated by a given actor, as well as other objectives deduced from other means the diagram is built in such a way that a main goal is being represented in more specific objectives until it may be measured with exact units. In this point, the objective is defined by a set of criteria, providing some measurable requirements to establish solutions.
- Ultimately, the goal tree of a given actor may look like the one below (in Spanish), in which is represented the problem statement and the objectives are spitted into specific objectives in lower levels until the lowest level, in which the objectives are expressed in concrete units, bringing a set of criteria.

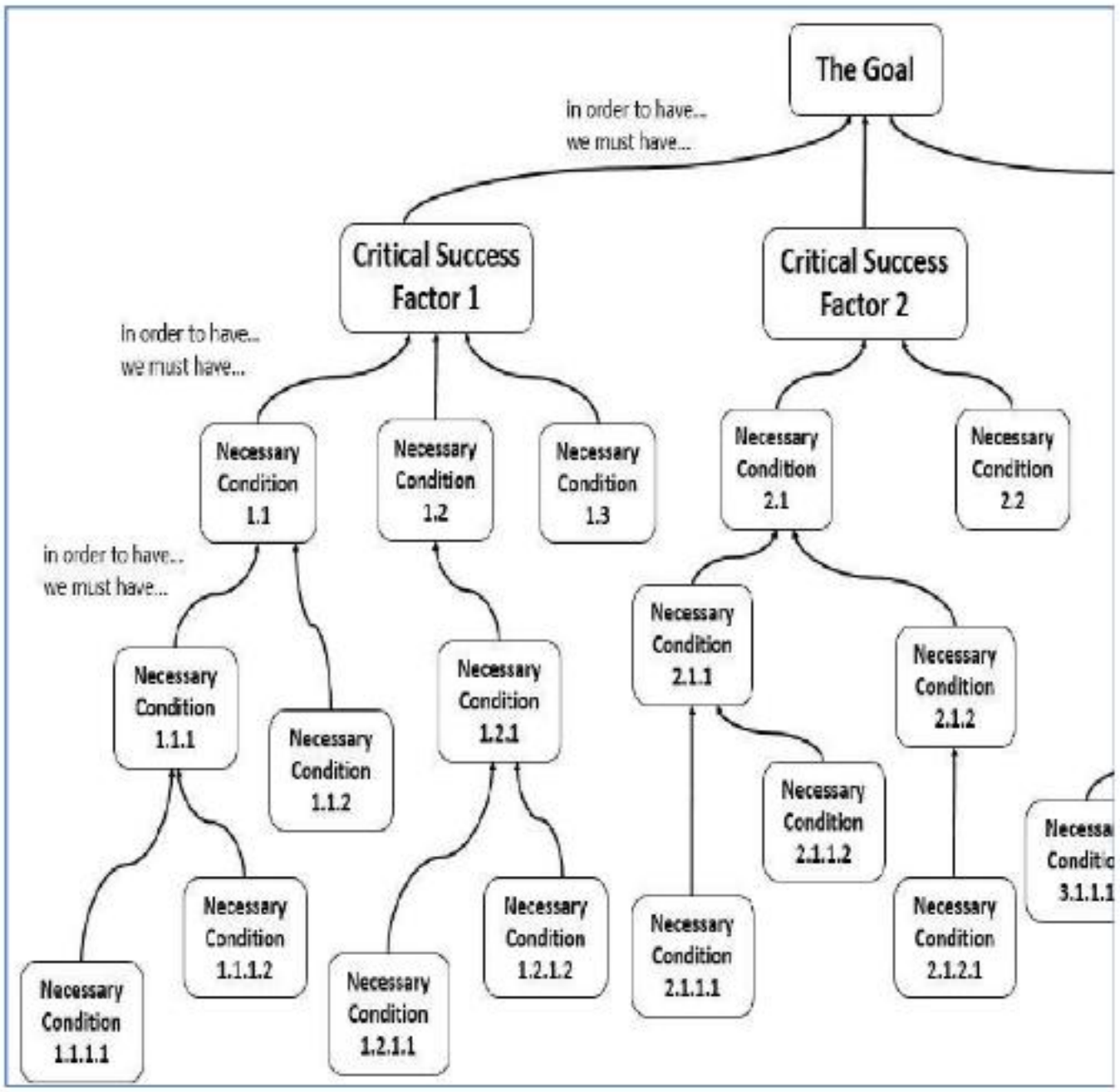

Fig - A Goal Trees

1.2 Rule Based Systems

- Rule-based systems can be used to create software that will provide an answer to a problem in place of a human expert. This type of system may also be called an expert system. Rule-based systems are also used in AI (artificial intelligence) programming and systems.

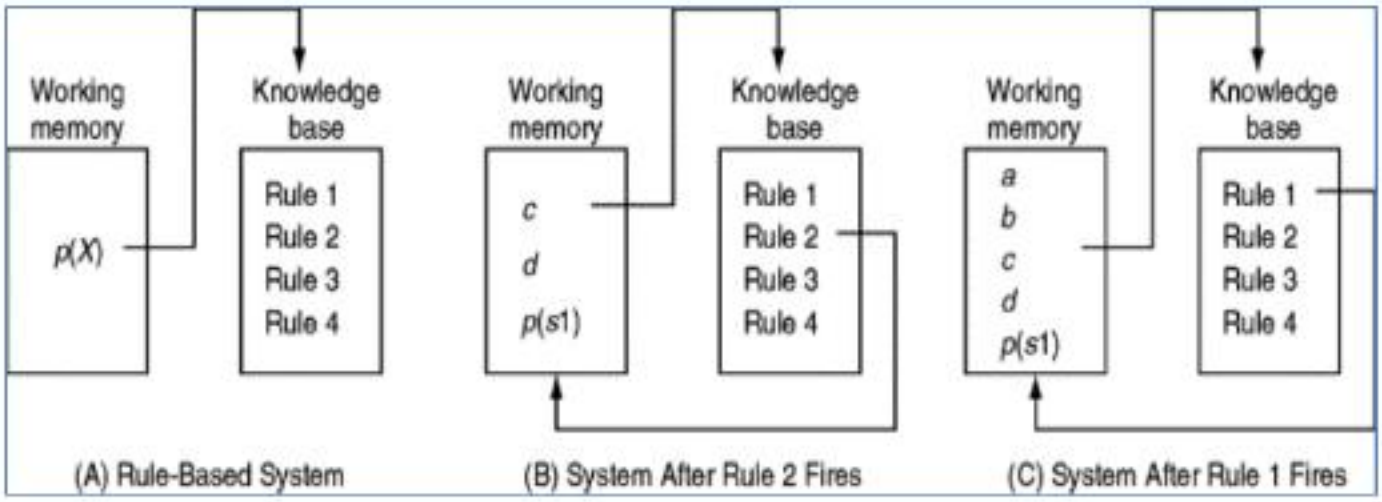

Fig - Rule-based systems

- In a rule-based system, the inference engine usually goes through a simple recognize-assert cycle. The control scheme is called forward chaining for data-driven reasoning, and backward chaining for goal-driven reasoning. The basic idea of forward chaining is when the premises of a rule are satisfied by the data, the expert system asserts the conclusions of the rule (the then portion) as true.

- A forward-chaining reasoning system starts by placing initial data in its working memory. Then the system goes through a cycle of matching the premises of rules with the facts in the working memory, selecting one rule, and placing its conclusion in the working memory.

- The control of the previous backward-chaining process performs a depth-first search, in which each new sub goal is searched exhaustively first before moving onto old sub goals. Other search strategies, such as breadth-first search, can also be applied.

- Given the same set of rules, forward chaining can also be applied to derive new conclusions from given data. For example, the algorithm of forward chaining with breadth-first search is as follows: Compare the content of the working memory with the premises of each rule in the rule base using the ordering of the rules.

- If the data in the working memory match a rule's premises, the conclusion is placed in the working memory, and the control moves to the next rule. Once all

rules have been considered, the control starts again from the beginning of the rule sets.

A typical rule-based system has four basic components

- A list of rules or rule base, which is a specific type of knowledge base
  An inference engine or semantic reasoned which infers information or takes action based on the interaction of input and the rule base. The interpreter executes a production system program by performing the following match-resolve-act cycle.

➢ Match: In this first phase, the left-hand sides of all productions are matched against the contents of working memory. As a result a conflict set is obtained, which consists of instantiations of all satisfied productions. An instantiation of a production is an ordered list of working memory elements that satisfies the left-hand side of the production.

➢ Conflict-Resolution: In this second phase, one of the production instantiations in the conflict set is chosen for execution. If no productions are satisfied, the interpreter halts.

➢ Act: In this third phase, the actions of the production selected in the conflict-resolution phase are executed. These actions may change the contents of working memory. At the end of this phase, execution returns to the first phase.

Temporary Working memory

A User interface or other connection to the outside world through which input and output signals are received and sent.

Applications

- A classic example of a rule-based system is the domain-specific expert system that uses rules to make deductions or choices. For example, an expert system might help a doctor choose the correct diagnosis based on a cluster of symptoms, or select tactical moves to play a game.
- Rule-based systems can be used to perform lexical analysis to compile or interpret computer programs, or in natural language processing.
- Rule-based programming attempts to derive execution instructions from a starting set of data and rules. This is a more indirect method than that employed by an imperative programming language, which lists execution steps sequentially.

1.3 Rule Based Expert Systems

Rules as a Knowledge Representation technique „

- Structure of a rule-based expert system Characteristics of an expert system „
- Forward/Backward chaining Conflict Resolution.
- Rules as a knowledge representation technique „

- The human mental process is internal; it is too complex to be represented as an algorithm. However, most experts are capable of expressing their knowledge in the form of rules for problem solving IF -THEN structure – relates given information or facts in the IF part to some action in the THEN part (description of how to solve a problem).
- Any rule consists of two parts: ...the IF part (antecedent – premise or condition) the THEN part (consequent – conclusion or action) „ A rule have multiple antecedents joined by AND (conjunction) or OR (disjunction). The antecedent of a rule incorporates: the object and its value are linked by an operator

Structure of Rule Based Expert Systems

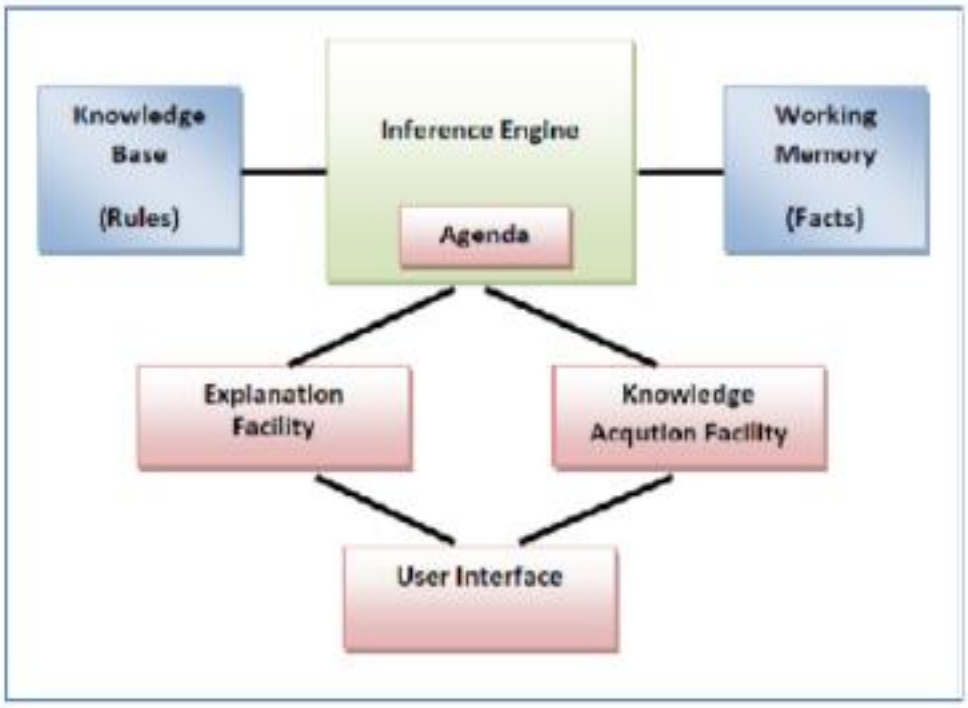

Fig - Structure of Rule Based Expert Systems

- The Knowledge base – contains the domain knowledge useful for problem solving. Each rule specifies a relation, recommendation, directive, strategy, and heuristics.
- IF (condition) THEN (action) – When the condition part of the rule is satisfied, the rule is said to fire and the action part is executed. „ The Database includes a

set of facts used to match against the IF (condition) parts of the rules stored in the knowledge base. „

- The Inference Engine caries out the reasoning whereby the expert system reaches a solution. It links the rules given in the knowledge base with the facts provided in the database.
- The explanation facilities enable the user to ask the expert system how a particular condition is reached and why a specific fact is needed. An expert system must be able to explain its reasoning and justify its advice, analysis or conclusion. „
- The user interface is the means of communication between a user seeking a solution and an expert system.

Characteristics of an expert system „

High-quality performance (speed of reaching a solution) - built to perform at a human expert level in a narrow, specialized domain. Apply heuristics to guide the reasoning and thus reduce the search area for a solution „ Explanation capability – enable the experts to review its own reasoning and explain its decisions „ Employ symbolic reasoning when solving a problem (symbol – used to represent different types of knowledge such as facts, rules, concepts)

2. Planning:

- The planning in Artificial Intelligence is about the decision making tasks performed by the robots or computer programs to achieve a specific goal.
- The execution of planning is about choosing a sequence of actions with a high likelihood to complete the specific task.

Blocks-World planning problem

- The blocks-world problem is known as Suss man Anomaly.
- No interleaved planners of the early 1970s were unable to solve this problem, hence it is considered as anomalous.
- When two sub goals G1 and G2 are given, a non interleaved planner produces either a plan for G1 concatenated with a plan for G2, or vice-versa.
- In blocks-world problem, three blocks labelled as 'A', 'B', 'C' are allowed to rest on the flat surface. The given condition is that only one block can be moved at a time to achieve the goal.
- The start state and goal state are shown in the following diagram.

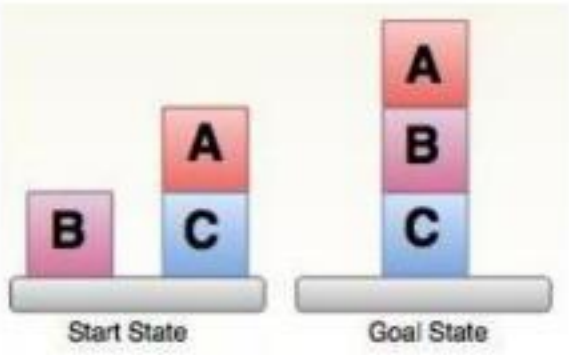

Fig – Blocks world planning problem

Components of Planning System

➢ **The planning consists of following important steps:**

- Choose the best rule for applying the next rule based on the best available heuristics.
- Apply the chosen rule for computing the new problem state.
- Detect when a solution has been found.
- Detect dead ends so that they can be abandoned and the system's effort is directed in more fruitful directions.
- Detect when an almost correct solution has been found.

➢ **Goal stack planning**

This is one of the most important planning algorithms, which is specifically used by STRIPS.

- The stack is used in an algorithm to hold the action and satisfy the goal. A knowledge base is used to hold the current state, actions.
- Goal stack is similar to a node in a search tree, where the branches are created if there is a choice of an action.

➢ **The important steps of the algorithm are as stated below:**

Start by pushing the original goal on the stack. Repeat this until the stack becomes empty. If stack top is a compound goal, then push its unsatisfied sub goals on the stack.
If stack top is a single unsatisfied goal then, replace it by an action and push the action's precondition on the stack to satisfy the condition.
If stack top is an action, pop it from the stack, execute it and change the knowledge base by the effects of the action.

If stack top is a satisfied goal, pop it from the stack.

2.1 STRIPS

The term strips has various meanings:

- A financial option composed of one call option and two put options with the same strike price
- A treasury security acronym for Separate Trading of Registered Interest and Principal of Securities, which are the securities obtained when trading the coupons and principal of bonds separately
- The Stanford Research Institute Problem Solver (STRIPS) is an automated planning technique that works by executing a domain and problem to find a goal. With STRIPS, you first describe the world. You do this by providing objects, actions, preconditions, and effects. These are all the types of things you can do in the game world.
- Once the world is described, you then provide a problem set. A problem consists of an initial state and a goal condition. STRIPS can then search all possible states, starting from the initial one, executing various actions, until it reaches the goal.
- A common language for writing STRIPS domain and problem sets is the Planning Domain Definition Language (PDDL). PDDL lets you write most of the code with English words, so that it can be clearly read and (hopefully) well understood. It's a relatively easy approach to writing simple AI planning problems.
- A lot of different problems can be solved using STRIPS and PDDL. As long as the world domain and problem can be described with a finite set of actions, preconditions, and effects, you can write a PDDL domain and problem to solve it.
- For example, stacking blocks, Rubik's cube, navigating a robot in Shake's, Star craft build orders, and a lot more, can be described using STRIPS and PDDL.
- The STRIPS representation is based on the idea that most things are not affected by a single action. For each action, STRIPS models when the action is possible and what primitive features are affected by the action.
- The effect of the action relies on the STRIPS assumption: All of the primitive features not mentioned in the description of the action stay unchanged.

The STRIPS representation for an action consists of

- the precondition, which is a set of assignments of values to features that must be true for the action to occur, and
- the effect, which is a set of resulting assignments of values to those primitive features that change as the result of the action

2.2 Forward and Backward State Space Planning

Forward State-Space Search:

- Planning with forward state-space search is similar to the problem-solving approach. It is sometimes called progression planning, because it moves in the forward direction.
- We start with the problem's initial state, considering sequences of actions until we reach a goal state.

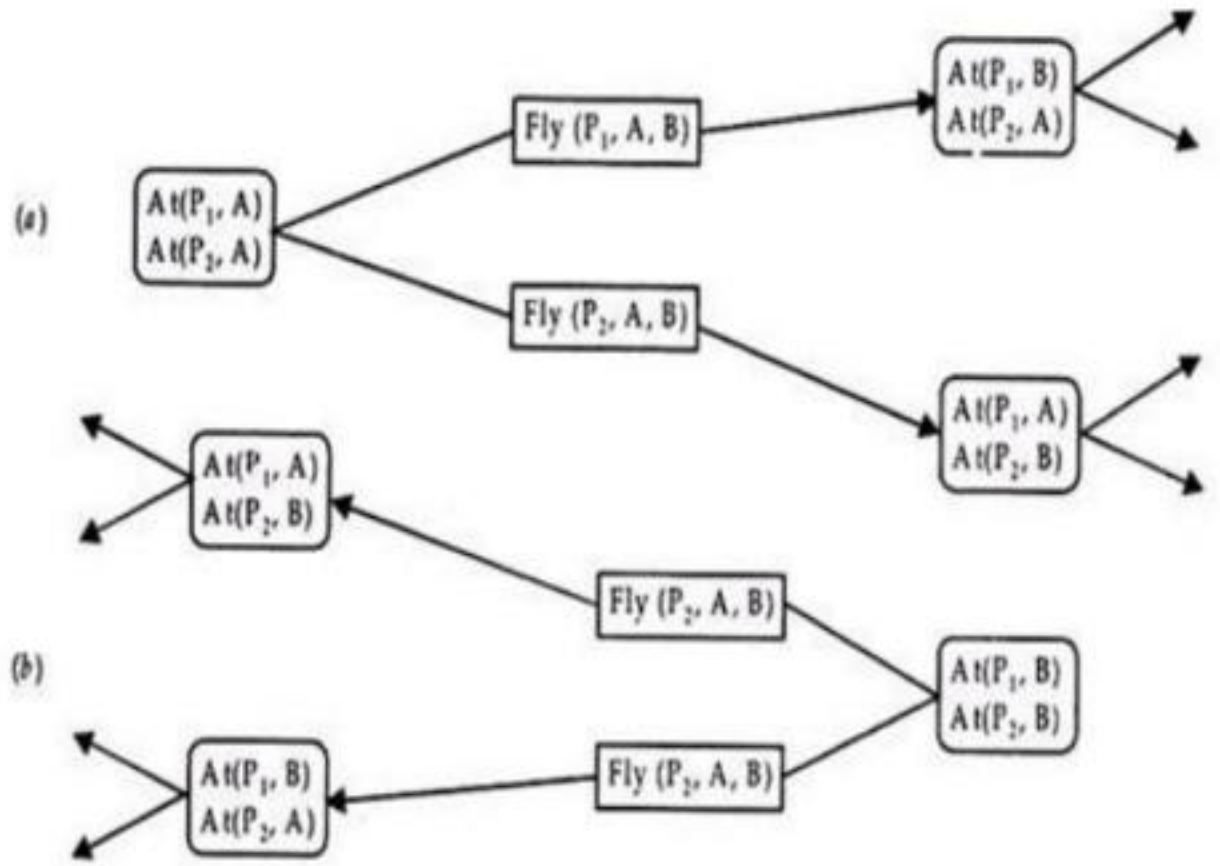

Fig – a) Froward state space search b) Backward state space search

The formulation of planning problem as state-space search problems is as follows:

- The initial state of the search is the initial state from the planning problem. In general each state will be set of positive ground literals; literals not appearing are false.
- The actions which are applicable to a state are all those whose preconditions are satisfied. The successor state resulting from an action is generated by adding the positive effect literals and deleting the negative effect literals.
- The goal test checks whether the state satisfies the goal of the planning problem.
- The step cost of each action is typically 1. Although it would be easy to allow different costs for different actions, this was seldom done by STRIPS planners.
- Since function symbols are not present, the state space of a planning problem is finite and therefore, any graph search algorithm such as A * will be a complete planning algorithm.
- From the early days of planning research it is known that forward state-space search is too inefficient to be practical. Mainly, this is because of a big branching factor since forward search does not address only relevant actions, (all applicable actions are considered). Consider for example, an air cargo problem with 10 airports, where each airport has 5 planes and 20 pieces of cargo.
- The goal is to move all the cargo at airport A to airport B. There is a simple solution to the problem: load the 20 pieces of cargo into one of the planes at A, fly the plane to B, and unload the cargo. But finding the solution can be difficult because the average branching factor is huge: each of the 50 planes can fly to 9 other airports, and each of the 200 packages can be either unloaded (if it is loaded), or loaded into any plane at its airport (if it is unloaded).
- On average, let's say there are about 1000 possible actions, so the search tree up to the depth of the obvious solution has about 1000 nodes. It is thus clear that a very accurate heuristic will be needed to make this kind of search efficient.

Backward State-Space Search:

- Backward search can be difficult to implement when the goal states are described by a set of constraints which are not listed explicitly. In particular, it is not always obvious how to generate a description of the possible predecessors of the set of goal states.
- The STRIPS representation makes this quite easy because sets of states can be described by the literals which must be true in those states.
- The main advantage of backward search is that it allows us to consider only relevant actions. An action is relevant to a conjunctive goal if it achieves one of the conjuncts of the goal. For example, the goal in our 10-airport air cargo problem is to have 20 pieces of cargo at airport B, or more precisely.

At (C_1 B) ∧ At (C_2 B)..... At (C_{20}, B)

Now consider the conjunct At (C_1, B). Working backwards, we can seek those actions which have this as an effect,

There is only one:

Unload (C_1p, B),

Where plane p is unspecified.

- We may note that there are many irrelevant actions which can also lead to a goal state. For example, we can fly an empty plane from Mumbai to Chennai; this action reaches a goal state from a predecessor state in which the plane is at Mumbai and all the goal conjuncts are satisfied.
- A backward search which allows irrelevant actions will still be complete, but it will be much less efficient. If a solution exists, it should be found by a backward search which allows only relevant action.
- This restriction to relevant actions only means that backward search often has a much lower branching factor than forward search.
- For example, our air cargo problem has about 1000 actions leading forward from the initial state, but only 20 actions working backward from the goal. Hence backward search is more efficient than forward searching.

Searching backwards is also called regression planning.

2.3 Goal Stack Planning

Basic Idea to handle interactive compound goals uses goal stacks, Here the stack contains:

- goals
- operators -- ADD, DELETE and PREREQUISITE lists
- A database maintaining the current situation for each operator used.
- Consider the following where wish to proceed from the start to goal state.

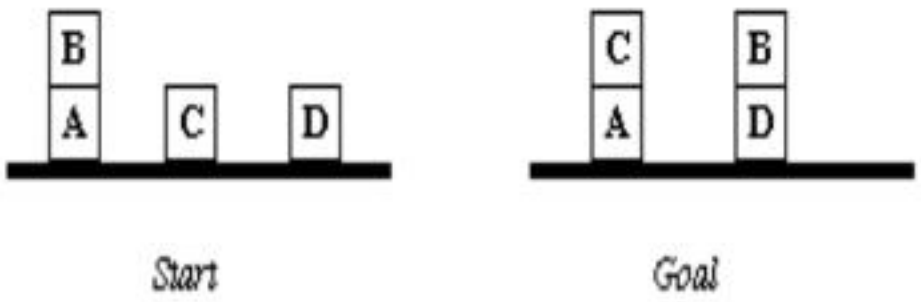

Fig - Goal Stack Planning Example

We can describe the start state:

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

And goal state:

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

- Initially the goal stack is the goal state.
- We then split the problem into four sub problems
- Two are solved as they already are true in the initial state -- ONTABLE (A), ONTABLE (D).

The Goal-Stack Planning Algorithm

We are currently considering STRIPS, the earliest type of AI planner. It did not follow all three of the key ideas of planning listed above, but it did represent states as logical sentences and decompose problems.

Input:

Initial-state I

Goals g1, g2, gn

Operator set O

Output: success or failure

Local state:

Problem-stack

World-state

The algorithm:

- world-state = I
- Push achieves (g1... gn) onto problem-stack.
- Loop:

If problem-stack is empty, return SUCCESS.
N = pop (problem-stack)
If N = achieve (g1... gn) then
If N is true in the world-state, then go to 3.
If N has previously been attempted, then return FAILURE.
Mark N as attempted.

Push N back on problem-stack.
Order the gi.
Push achieve (gi) on problem-stack in order.
Else if N = achieve (g) then
If N is true in the world-state, then go to 3.
Choose an operator from o from O that possibly adds g. If none exists, fail.
Choose a set of variable bindings that grounds o and makes o add g.
Push apply (o) onto problem-stack.
Push achieve (precondition (o)) onto the problem-stack.
Else if N = apply (o)
world-state = execute (o, world-state)

2.4 Plan Space Planning

In state-space planning, a program searches through a space of world states, seeking to find a path or paths that will take it from its initial state to a goal state. State-space planning is too inflexible, because:

- It creates plans that are total orderings of a set of steps, and
- It assembles these plans in exactly the same order.

Plan-Space Planning Redux

In plan-space planning, a program searches through a space of plans, seeking a plan that will take it from its initial state to a goal state. In this approach, we redefine some of the terms of our search:

- A plan is a set of steps and a set of constraints on the ordering of the steps.
- A state is a plan.
- The goal state is a plan that achieves all specified goals.
- An operator creates a new plan from an old plan.

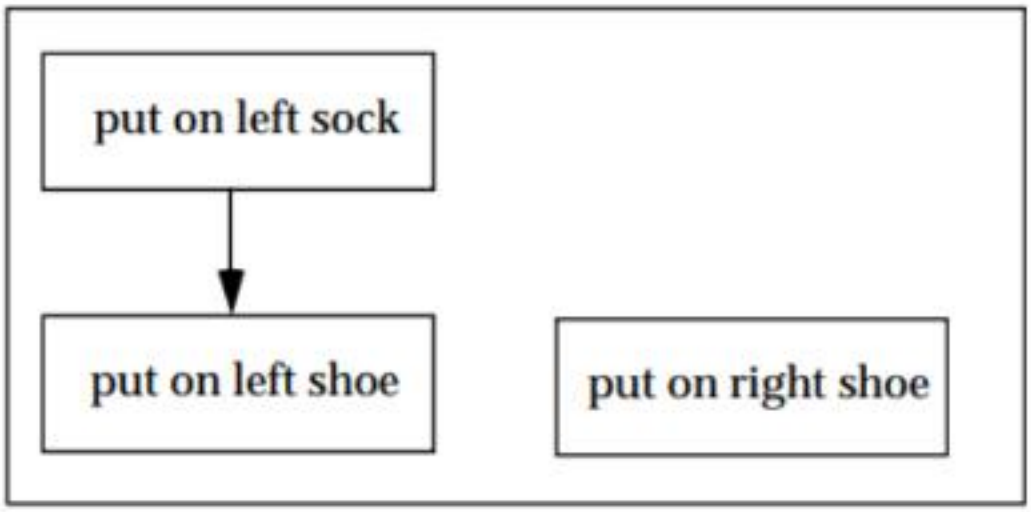

Fig - Plan Space Planning

2.5 A Unified Framework for Planning

- Users of AI systems may rely upon them to produce plans for achieving desired objectives. Such AI systems should be able to compute obfuscated plans whose execution in adversarial situations protects privacy, as well as legible plans which are easy for team members to understand in cooperative situations.
- We develop a unified framework that addresses these dual problems by computing plans with a desired level of comprehensibility from the point of view of a partially informed observer.
- For adversarial settings, our approach produces obfuscated plans with observations that are consistent with at least k goals from a set of decoy goals. By slightly varying our framework, we present an approach for goal legibility in cooperative settings which produces plans that achieve a goal while being consistent with at most j goals from a set of confounding goals.
- In addition, we show how the observability of the observer can be controlled to either obfuscate or clarify the next actions in a plan when the goal is known to the observer. We present theoretical results on the complexity analysis of our problems.
- A robust intelligent agent must have three general characteristics. First, it should be able to plan, to generate possible action sequences that lead to the achievement of goals.
- Second, the agent should learn from its problem-solving experience in a domain, improving its ability from previous attempts at plan generation. Finally, the agent should integrate planning and learning with other aspects of behavior, such as execution and perception.
- These capabilities are central to human behavior, and we believe they are essential to the success of any agent that is situated in a complex physical environment. Our long-term goal is to develop a unified architecture that

provides practical abilities of this sort while remaining consistent with knowledge of human cognition.

3. Constraint Satisfaction:

- Look up Constraint Satisfaction Problems (CSPs) and depth-first search (DFS). Wikipedia's fine, but a very good source is the CSC242 text, Russell and Norvig's book Artificial Intelligence, a Modern Approach.
- The relevant pages are on E-reserve for this class, available through Blackboard: look for reading "Russell: Constraint Satisfaction."
- CSPs are a hugely important class of practical problems (e.g. classroom and job scheduling, military logistics, Sudoku ...) and there are a number of more or less ingenious ways to solve them. To solve a CSP you must assign a value to each of a set of variables so that no constraints are violated.
- Favourite examples are crypt arithmetic puzzles like SEND + MORE = MONEY. Here the variables are the letters S, E, N, D, M, O, R, Y, possible values are the integers 0,...,9, and there are constraints like each variable represents a single integer, and the sum (in general, arithmetic operation) must work out after substituting the integer values for the variables.
- Other examples are map colouring, where the variables are regions, the values are colors, and the constraint is that no adjacent regions can have the same color, which means respecting the specific instance-dependent constraints induced by the neighbor relations in the map.

3.1 N-Queens

- We'll illustrate constraint programming (CP) by a combinatorial problem based on the game of chess. In chess, a queen can attack horizontally, vertically, and diagonally. The N-queens problem asks:
- How can N queens be placed on an NxN chessboard so that no two of them attack each other?

Below, you can see one possible solution to the N-queens problem for N = 4.

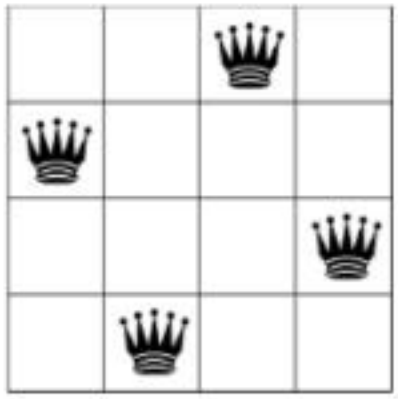

Fig – Representation of N-Queens

No two queens are on the same row, column, or diagonal.

- Note that this isn't an optimization problem: we want to find all possible solutions, rather than one optimal solution, which make it a natural candidate for constraint programming.
- The following sections describe the CP approach to the N-queens problem, and present Python programs that solve it using both the CP-SAT solver and the original CP solver.

CP approach to the N-queens problem

- A CP solver works by systematically trying all possible assignments of values to the variables in a problem, to find the feasible solutions. In the 4-queens problem, the solver starts at the leftmost column and successively places one queen in each column, at a location that is not attacked by any previously placed queens.

3.2 Constraint Propagation

- Constraint propagation is the process of communicating the domain reduction of a decision variable to all of the constraints that are stated over this variable. This process can result in more domain reductions. These domain reductions, in turn, are communicated to the appropriate constraints.
- This process continues until no more variable domains can be reduced or when a domain becomes empty and a failure occurs. An empty domain during the initial constraint propagation means that the model has no solution.
- For example, consider the decision variables y with an initial domain [0..10], z with an initial domain [0..10] and t with an initial domain [0..1], and the constraints

$y + 5*z <= 4$

$t != z$

$t != y$

Over these three variables:

- The domain reduction of the constraint y + 5*z <=4 reduces the domain of y to [0..4] and z to [0]. The variable z is thus fixed to a single value. Constraint propagation invokes domain reduction of every constraint involving z.
- Domain reduction is invoked again for the constraint y + 5*z <= 4, but the variable domains cannot be reduced further. Domain reduction of the constraint t != z is invoked again, and because z is fixed to 0, the constraint removes the value 0 from the domain of t.
- The variable t is now fixed to the value 1, and constraint propagation invokes domain reduction of every constraint involving t, namely t != z and t != y. The constraint that can reduce domains further is t != y. Domain reduction removes the value 1 from the domain of y.

Constraint propagation is performed on constraints involving y; however, no more domain reduction can be achieved and the final domains are:

- y = [0 2..4],
- z = [0] and
- t = [1].

3.3 Scene Labelling

- In the above example of discrete labelling regions can only be labelled in a definite manner, either a region is or is not a door. Scene labelling as a whole may be consistent or inconsistent.
- If a scene cannot be consistently labelled then we might define that scene as impossible, as for example in the case of impossible objects which are represented as line drawings in the AI textbooks. That is all very well, but in an imperfect world it is somewhat unrealistic to ask for certainty in a labelled scene.
- This leads to the idea of a probabilistic labelling, in which scene elements may be labelled with probability in a continuum (0.0 to 1.0); for example the set of probabilities for a given region in Figure 1, above, might be { Door=0.1, Bin=0.3, Floor=0.1, Wall=0.2, Ceiling=0.3 }.
- An algorithm for labelling the scene consistently should update these probabilities to obtain an optimum global labelling satisfying as far as possible the unary characteristics of the regions, and the N-ary characteristics of their relationships. We shall consider this further with the specific examples.xs of subsequent sections.

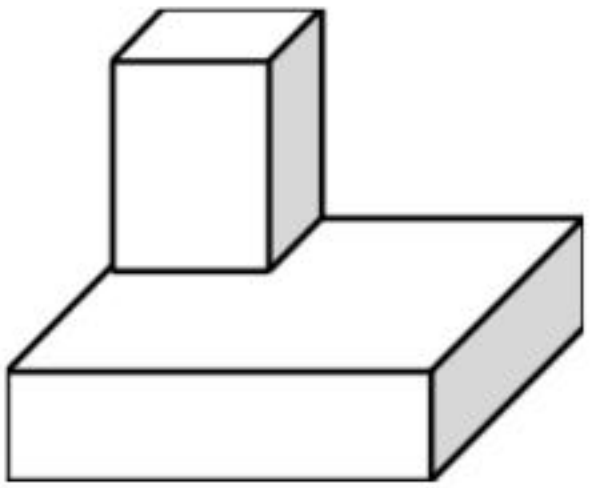

Fig – Architecture of Scene Labelling

- Before considering a number of approaches to object recognition by scene labelling, we introduce a few basic definitions. In general, the set of object models is defined,

$$U = \{O_1, O_2, ..., O_n\} \qquad (1)$$

- Each of the n object models may be represented by a set of m(i) primitives, where the parameter, i, denotes the model number (1<=i<=n). These primitives might be surfaces within a B-Rep model, space curves within a wire frame or B-Rep model, volumetric components in a CSG structure and so on.

$$O_i = \{f_1, f_2, ..., f_{m(i)}\} \qquad (2)$$

- Similarly, the segmented scene consists of a set of m(s) primitives, arising from segmentation of 3D range data or 2D intensity data for example, such that

$$O_s = \{g_1, g_2, ..., g_{m(s)}\} \qquad (3)$$

- The problem is to find the optimum injection mapping, $M_p : O_s \mapsto O_i$, from the set of scene features defined by Equation 3 to the set of model features defined by Equation 2 so that the defined metric for the mapping M_p is a maximum,

$$M_p \geq M_q \quad for\ all\ q \qquad (4)$$

Reference:

1. Stuart Russell and Peter Norvig, "Artificial Intelligence: A Modern Approach", Third edition, Pearson, 2003, ISBN :10: 0136042597
2. Michael Jenkin, Gregory, " Computational Principals of Mobile Robotics", Cambridge University Press, 2010, ISBN : 978-0-52-187157-0
3. Nilsson Nils J , "Artificial Intelligence: A new Synthesis, Morgan Kaufmann Publishers Inc. San Francisco, CA, ISBN: 978-1-55-860467-4
4. Patrick Henry Winston, "Artificial Intelligence", Addison-Wesley Publishing Company, ISBN: 0-201-53377-4
5. Andries P. Engelbrecht-Computational Intelligence: An Introduction, 2nd Edition-Wiley India- ISBN: 978-0-470-51250-0

Unit III Logic and Reasoning

1. Knowledge Based Reasoning:
    - 1.1 Agents
    - 1.2 Facets of Knowledge
2. Logic and Inferences:
    - 2.1 Formal Logic
    - 2.2 Propositional and First Order Logic
    - 2.3 Resolution in Propositional and First Order Logic
    - 2.4 Deductive Retrieval
    - 2.5 Backward Chaining
    - 2.6 Second order Logic
3. Knowledge Representation:
    - 3.1 Conceptual Dependency
    - 3.2 Frames
    - 3.3 Semantic nets

1. Knowledge Based Reasoning:

- In a conventional program, domain knowledge is intimately intertwined with software for controlling the application of that knowledge. In a knowledge-based system, the two roles are explicitly separated. In the simplest case there are two modules: the knowledge module is called the knowledge base and the control module is called the inference engine. Some interface capabilities are also required for a practical system, as shown in Fig.
- Within the knowledge base, the programmer expresses information about the problem to be solved. Often this information is declarative, i.e. the programmer states some facts, rules, or relationships without having to be concerned with the detail of how and when that information should be applied.
- These latter details are determined by the inference engine, which uses the knowledge base as a conventional program uses a data file. A KBS is analogous to the human brain, whose control processes are approximately unchanging in their nature, like the inference engine, even though individual behavior is continually modified by new knowledge and experience, like updating the knowledge base.
- As the knowledge is represented explicitly in the knowledge base, rather than implicitly within the structure of a program, it can be entered and updated with relative ease by domain experts who may not have any programming expertise.
- A knowledge engineer is someone who provides a bridge between the domain expertise and the computer implementation. The knowledge engineer may make use of meta-knowledge, i.e. knowledge about knowledge, to ensure an efficient implementation.

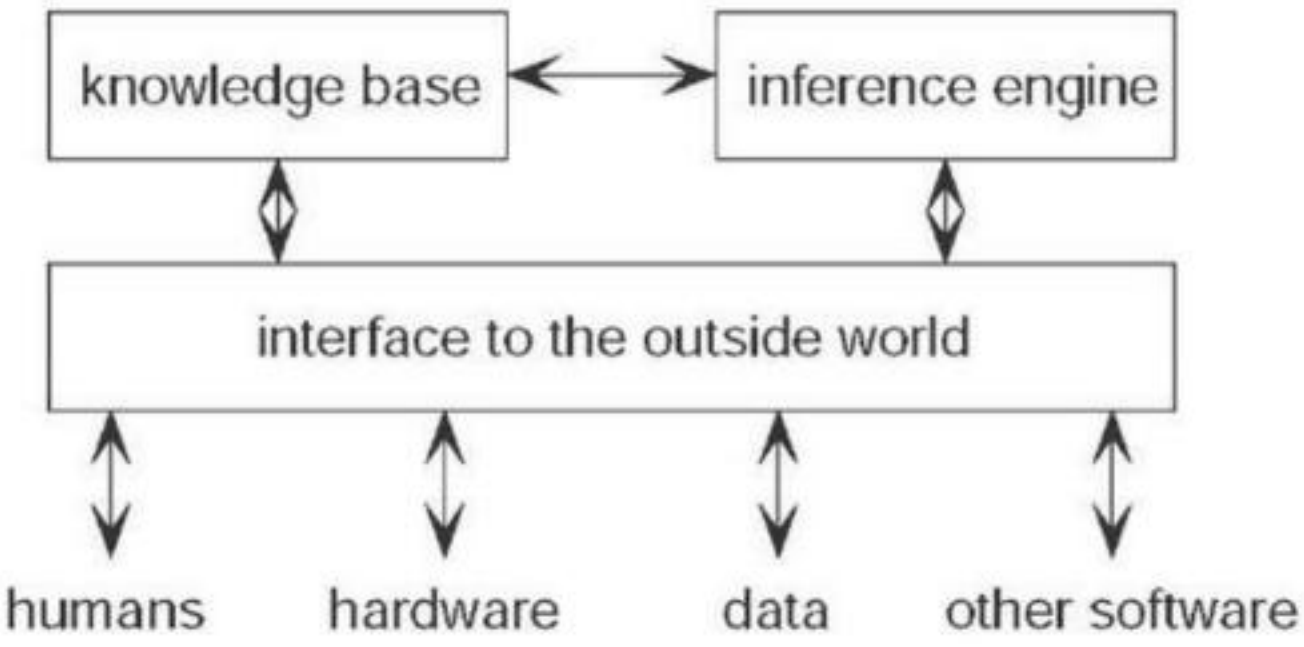

Fig - The main components of a knowledge-based system

- Traditional knowledge engineering is based on models of human concepts. However, it has recently been argued that animals and pre-linguistic children operate effectively in a complex world without necessarily using concepts.
- Moss has demonstrated that agents using non-conceptual reasoning can outperform stimulus-response agents in a grid-world test bed. These results may justify the building of non-conceptual models before moving on to conceptual ones.

1.1 Agents

- An intelligent agent needs knowledge about the real world for taking decisions and reasoning to act efficiently.
- Knowledge-based agents are those agents who have the capability of maintaining an internal state of knowledge, reason over that knowledge, update their knowledge after observations and take actions.
- These agents can represent the world with some formal representation and act intelligently. Knowledge-based agents are composed of two main parts:

➢ **Knowledge-base and Inference system**.

A knowledge-based agent must able to do the following:

- An agent should be able to represent states, actions, etc.
- An agent Should be able to incorporate new percepts
- An agent can update the internal representation of the world
- An agent can deduce the internal representation of the world
- An agent can deduce appropriate actions.

The architecture of knowledge-based agent:

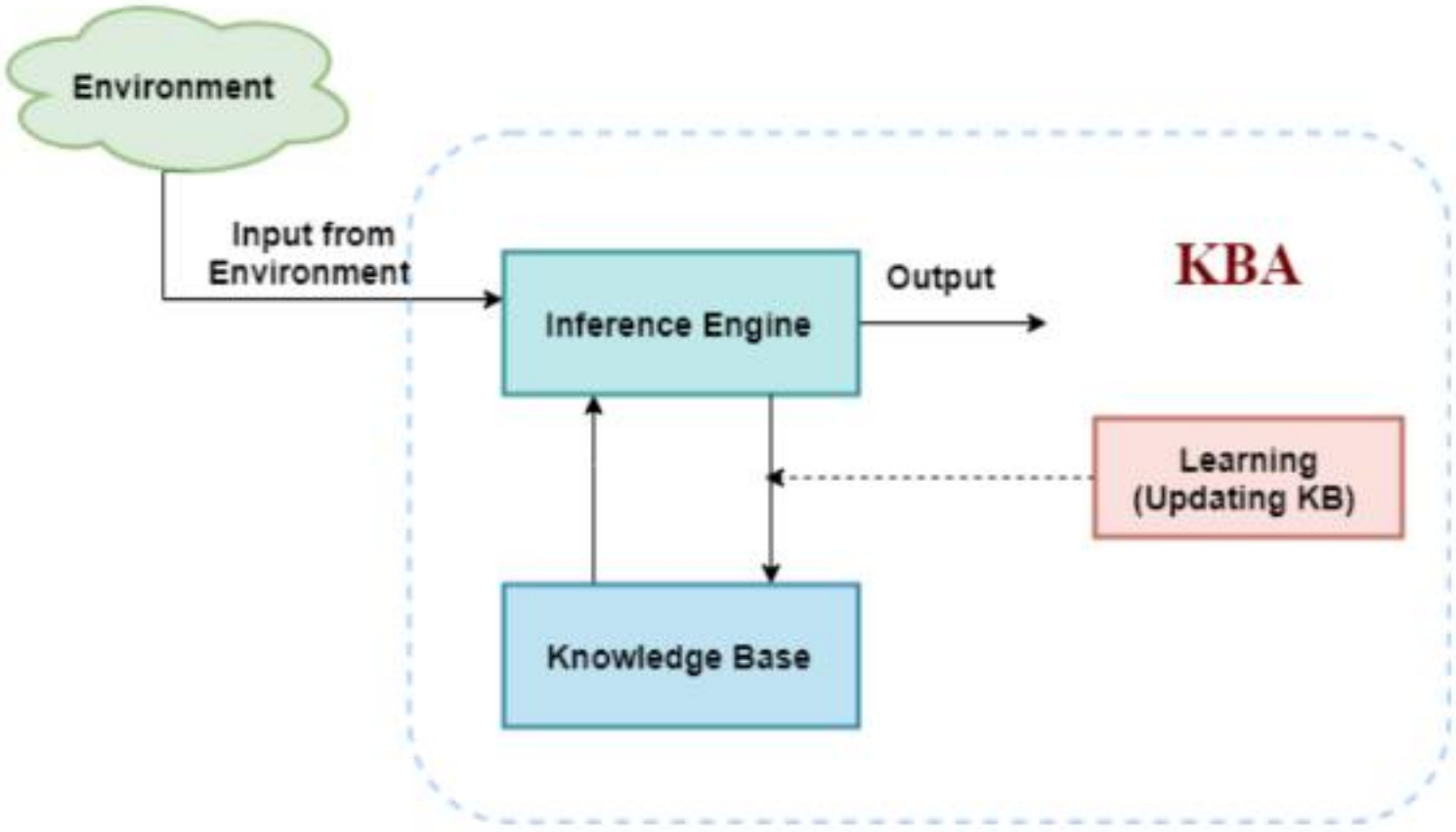

Fig - Architecture of knowledge-based agent

- The above diagram is representing a generalized architecture for a knowledge-based agent. The knowledge-based agent (KBA) takes input from the environment by perceiving the environment.
- The input is taken by the inference engine of the agent and which also communicate with KB to decide as per the knowledge store in KB. The learning element of KBA regularly updates the KB by learning new knowledge.

Knowledge base:

Knowledge-base is a central component of a knowledge-based agent, it is also known as KB. It is a collection of sentences. These sentences are expressed in a language which is called a knowledge representation language. The Knowledge-base of KBA stores fact about the world.

1.2 Facets of Knowledge

- A frame is a record like structure which consists of a collection of attributes and its values to describe an entity in the world. Frames are the AI data structure which divides knowledge into substructures by representing stereotypes situations.
- It consists of a collection of slots and slot values. These slots may be of any type and sizes. Slots have names and values which are called facets.
- **Facets:** The various aspects of a slot are known as Facets. Facets are features of frames which enable us to put constraints on the frames.

- **Example:** IF-NEEDED facts are called when data of any particular slot is needed. A frame may consist of any number of slots, and a slot may include any number of facets and facets may have any number of values.
- A frame is also known as slot-filter knowledge representation in artificial intelligence. Frames are derived from semantic networks and later evolved into our modern-day classes and objects.
- A single frame is not much useful. Frames system consists of a collection of frames which are connected. In the frame, knowledge about an object or event can be stored together in the knowledge base.
- The frame is a type of technology which is widely used in various applications including Natural language processing and machine visions.

2. Logic and Inferences:

- In artificial intelligence, we need intelligent computers which can create new logic from old logic or by evidence, so generating the conclusions from evidence and facts is termed as Inference.

Inference rules:

- Inference rules are the templates for generating valid arguments. Inference rules are applied to derive proofs in artificial intelligence, and the proof is a sequence of the conclusion that leads to the desired goal.
- In inference rules, the implication among all the connectives plays an important role. Following are some terminologies related to inference rules:

  - **Implication:** It is one of the logical connectives which can be represented as P $\rightarrow$ Q. It is a Boolean expression.
  - **Converse:** The converse of implication, which means the right-hand side proposition, goes to the left-hand side and vice-versa. It can be written as Q $\rightarrow$ P.
  - **Contra positive:** The negation of converse is termed as contrapositive, and it can be represented as $\neg$ Q $\rightarrow$ $\neg$ P.
  - **Inverse:** The negation of implication is called inverse. It can be represented as $\neg$ P $\rightarrow$ $\neg$ Q.

  From the above term some of the compound statements are equivalent to each other, which we can prove using truth table:

P	Q	P → Q	Q→ P	¬ Q → ¬ P	¬ P → ¬ Q.
T	T	T	T	T	T
T	F	F	T	F	T
F	T	T	F	T	F
F	F	T	T	T	T

Hence from the above truth table, we can prove that P ⟶ Q is equivalent to ¬ Q ⟶ ¬ P, and Q⟶ P is equivalent to ¬ P ⟶ ¬ Q.

2.1 Formal Logic

- The syntactical part of formal logic is that part which describes formal logic as a symbol system. Beginning with the definitions of logical language and logical formula, the syntactical part of formal logic proceeds to develop such notions as axiomatic theory, formal consequence, etc.
- A logical language is a subset of the words generated from some given alphabet. It should be made clear that formal logic is a generic term; that is, that there exist a number of logics (the study of which can be started by reading the book that McCauley [1981] wrote for linguists).
- Logics may differ in two ways: Two different logics need not share the same underlying logical language; however, if they do, the relations of formal consequence characterizing each of them must be different.
- Informally, the relation of formal consequence of a given logic determines the class of reasonings that arc to be allowed by the logic. In fact, since a formula is an expression of a logical language, it is natural to define a relation of formal consequence as a function which assigns a set of formulas (the conclusions) to another set of formulas.
- Much of the impetus for the development of mathematical logic came from the desire of providing a solid foundation for mathematics. Nowadays it is computer science that has taken the leading role and it will be the applications and needs in this area that are bound to guide the future of formal logic.

2.2 Propositional and First Order Logic

- Propositional logic is the simplest logic illustrates basic ideas using propositions P1, Snow is Whyte P2, Today it is raining P3, This automated reasoning course is boring Pi is an atom or atomic formula Each Pi can be either true or false but never both The values true or false assigned to each proposition is called truth value of the proposition.
- Recursive dentition of well-formed formulas 1 An atom is a formula 2 If S is a formula, ¬S is a formula (negation) 3 If S1 and S2 are formulas, S1 ∧ S2 is a

formula (conjunction) 4 If S1 and S2 are formulas, S1 ∨ S2 is a formula (disjunction) 5 All well-formed formulas are generated by applying above rules Shortcuts: S1 → S2 can be written as ¬S1 ∨ S2 S1 ↔ S2 can be written as (S1 → S2) ∧ (S2 → S1)

Relationships between truth values of atoms and truth values of formulas

$\neg S$	is true iff	S	is false		
$S_1 \wedge S_2$	is true iff	S_1	is true **and**	S_2	is true
$S_1 \vee S_2$	is true iff	S_1	is true **or**	S_2	is true
$S_1 \rightarrow S_2$	is true iff	S_1	is false **or**	S_2	is true
i.e.,	is false iff	S_1	is true **and**	S_2	is false
$S_1 \leftrightarrow S_2$	is true iff	$S_1 \rightarrow S_2$	is true **and**	$S_2 \rightarrow S_1$	is true

- FOL is not decidable to prove that a formula is valid in FOL we cannot simply enumerate all its possible interpretations possible interpretations of a formula can be innately many: we can have an innate number of domains. We need an automated mechanism to verify inconsistent formulas.

Evaluation of FOL formulas

Given an interpretation $I = \langle D, A \rangle$, FOL formulas are evaluated to **true** or **false** according to the following rules:

- If S is an atomic formula and $S \triangleq P(t_1, \cdots, t_n)$, S is **true** iff $P^A(t_1^A, \cdots, t_n^A) = \top$
- If S is an atomic formula and $S \triangleq t_1 = t_2$, S is **true** iff $t_1^A = t_2^A$.
- If S is a formula evaluated to **true** then $\neg S$ is **false**.
- If S and T are two formulas then $S \wedge T$ is **true** iff A and T are **true**.
- If S and T are two formulas then $S \vee T$ is **true** iff A or T are **true**
- If $S \triangleq \forall x G$ is **true** iff G is true for every element $d \in D$.
- If $S \triangleq \exists x G$ is **true** iff G is true for at least one element $d \in D$.

- Given an interpretation I = hD, Ai, and FOL formulas are evaluated to true or false according to the following rules:
- If S is an atomic formula and S , P(t1, · · ,tn), S is true iP A(t A 1 , · · ,t A n) = > If S is an atomic formula and S , t1 = t2, S is true i t A 1 = t A 2 . If S is a formula evaluated to true then ¬S is false. If S and T are two formulas then S ∧ T is true i A and T are true. If S and T are two formulas then S ∨ T is true i A or T are true If S , ∀xG is true i G is true for every element d ∈ D. If S, ∃xG is true i G is true for at least one element d ∈ D.

First-Order logic:

- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in a concise way.
- First-order logic is also known as Predicate logic or First-order predicate logic. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.
- First-order logic (like natural language) does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:
- **Objects:** A, B, people, numbers, colors, wars, theories, squares, pits, rumpus.
- **Relations:** It can be unary relation such as: red, round, is adjacent, or n-any relation such as: the sister of, brother of, has color, comes between
- **Function:** Father of, best friend, third inning of, end of.

2.3 Resolution in Propositional and First Order Logic

Resolution is a valid inference rule producing a new clause implied by two clauses containing complementary literals – A literal is an atomic symbol or its negation, i.e., P, ~P

• Amazingly, this is the only interference rule you need to build a sound and complete theorem proverb – Based on proof by contradiction and usually called resolution refutation

• The resolution rule was discovered by Alan Robinson (CS, U. of Syracuse) in the mid 60s. Resolution is a sound and complete inference procedure for unrestricted FOL

First Order Logic

- In first-order logic, a predicate can only refer to a single subject. First-order logic is also known as first-order predicate calculus or first-order functional calculus.
- A sentence in first-order logic is written in the form Px or P(x), where P is the predicate and x is the subject, represented as a variable. Complete sentences are logically combined and manipulated according to the same rules as those used in Boolean algebra.
- First-order logic is symbolized reasoning in which each sentence, or statement, is broken down into a subject and a predicate. The predicate modifies or defines the properties of the subject.
- In first-order logic, a sentence can be structured using the universal quantifier (symbolized $\forall$) or the existential quantifier ($\exists$). Consider a subject that is a variable represented by x. Let A be a predicate "is an apple," F be a predicate "is a fruit," S be a predicate "is sour"', and M be a predicate "is mushy." Then we can say

  $\forall X: Ax \Rightarrow FX$

  Which translates to "For all x, if x is an apple, then x is a fruit." We can also say such things as

  $\exists x : Fx \Rightarrow Ax$

  $\exists x : Ax \Rightarrow Sx$

  $\exists x: Ax \Rightarrow Mx$

  Where the existential quantifier translates as "For some."

- First-order logic can be useful in the creation of computer programs. It is also of interest to researchers in artificial intelligence (AI). There are more powerful forms of logic, but first-order logic is adequate for most everyday reasoning.
- The Incompleteness Theorem , proven in 1930, demonstrates that first-order logic is in general undecidable. That means there exist statements in this logic form that, under certain conditions, cannot be proven either true or false

2.4 Deductive Retrieval

- Deductive retrieval is a generalization of simple data retrieval. Though the extensions appear relatively small, the result is in fact a full-fledged alternative to the declarative approach seen in C++, Java, PHP, ..., and the functional approach seen in Lisp, Scheme, Haskell,.... This alternative is called declarative programming.
- A data retriever has a database of assertions, e.g., (married john Mary), and so on. The main operations are storing assertions in the database retrieving all assertions that match a query pattern.
- The retriever looks through the database and returns all the assertions that match the input query pattern. Assertions are usually indexed and organized for fast retrieval, but the semantics is as if all assertions are matched against the query.

A deductive retriever adds the following operations:

- storing rules in the database
- using those rules to infer answers to queries, in addition to the assertions explicitly stored
- Deductive retrieval has these major processes:
- unification -- generalized pattern matching
- backward chaining -- linking rules together to form a conclusion
- backtracking -- trying all possibilities until one or all answers are found

2.5 Backward Chaining

- Backward chaining is the logical process of inferring unknown truths from known conclusions by moving backward from a solution to determine the initial conditions and rules.
- Backward chaining is often applied in artificial intelligence (AI) and may be used along with its counterpart, forward chaining.
- In AI, backward chaining is used to find the conditions and rules by which a logical result or conclusion was reached. An AI might utilize backward chaining to find information related to conclusions or solutions in reverse engineering or game theory applications.
- Backward chaining is used in automated theorem proving tools, inference engines, proof assistants and other artificial intelligence applications.
- As a goal-driven and top-down form of reasoning, backward chaining usually employs a depth-first search strategy by starting from a conclusion, result or goal and going backward to infer the conditions from which it resulted.

Backward chaining traces back through the code, for example, and look through a rules table.

- In the rules table, it seeks out any actions that are specified in if-then statements, applying logic to determine which of the possible actions would have caused the end result.
- Backward chaining and its opposite, forward chaining, use deductive reasoning. Forward chaining is used to break down the logic sequence and work through it from beginning to end by attaching each step after the previous one is solved.

2.6 Second order Logic

- In logic and mathematics second-order logic is an extension of first-order logic, which itself is an extension of propositional logic. Second-order logic is in turn extended by higher-order logic and type theory.
- First-order logic uses only variables that range over individuals (elements of the domain of discourse); second-order logic has these variables as well as additional variables that range over sets of individuals.
- For example, the second-order sentence $\forall P\, \forall x\, (x \in P \vee x \notin P)$ says that for every set P of individuals and every individual x, either x is in P or it is not (this is the principle of bivalence).
- Second-order logic also includes variables quantifying over functions, and other variables as explained in the section Syntax below. Both first-order and second-order logic use the idea of a domain of discourse. The domain is a set of individual elements which can be quantified over.

Expressive power of second order logic

- Second-order logic is more expressive than first-order logic. For example, if the domain is the set of all real numbers, one can assert in first-order logic the existence of an additive inverse of each real number by writing $\forall x\, \exists y\, (x + y = 0)$ but one needs second-order logic to assert the least-upper-bound property for sets of real numbers, which states that every bounded, nonempty set of real numbers has a supermom.
- If the domain is the set of all real numbers, the following second-order sentence expresses the least upper bound property:

$$\forall A\left[(\exists w(w \in A) \wedge \exists z\, \forall w(w \in A \rightarrow w \leq z)) \rightarrow \exists x\, \forall y([\forall w(w \in A \rightarrow w \leq y)] \leftrightarrow x \leq y)\right].$$

- In second-order logic, it is possible to write formal sentences which say "the domain is finite" or "the domain is of countable cardinality." To say that the domain is finite, use the sentence that says that every injective function from the domain to itself is subjective.

- To say that the domain has countable cardinality, use the sentence that says that there is a bijection between every two infinite subsets of the domain. It follows from the upward Lowenheim Skolem theorem that it is not possible to characterize finiteness or countability in first-order logic.

Syntax

- The syntax of second-order logic tells which expressions are well formed formulas. In addition to the syntax of first-order logic, second-order logic includes many new sorts (sometimes called types) of variables. These are:

- A sort of variables that range over sets of individuals. If S is a variable of this sort and t is a first-order term then the expression $t \in S$ (also written S (t) or St) is an atomic formula. Sets of individuals can also be viewed as unary relations on the domain.

- For each natural number k there is a sort of variable that ranges over all k-ary relations on the individuals. If R is such a k-ary relation variable and $t_1 \ldots t_k$ are first-order terms then the expression $R(t_1 \ldots t_k)$ is an atomic formula.

- For each natural number k there is a sort of variable that ranges over functions that take k elements of the domain and return a single element of the domain. If f is such a k-ary function symbol and $t_1, \ldots, t_k$ are first-order terms then the expression $f(t_1, \ldots, t_k)$ is a first-order term.

3. Knowledge Representation:

- Humans are best at understanding, reasoning, and interpreting knowledge. Human knows things, which is knowledge and as per their knowledge they perform various actions in the real world. But how machines do all these things comes under knowledge representation and reasoning. Hence we can describe Knowledge representation as following:

- Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which concerned with AI agents thinking and how thinking contributes to intelligent behavior of agents.

- It is responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve the complex real world problems such as diagnosis a medical condition or communicating with humans in natural language.

- It is also a way which describes how we can represent knowledge in artificial intelligence. Knowledge representation is not just storing data into some database, but it also enables an intelligent machine to learn from that knowledge and experiences so that it can behave intelligently like a human.

Fig - Various types of knowledge

Following are the kind of knowledge which needs to be represented in AI systems:

- **Object:** All the facts about objects in our world domain. E.g., Guitars contains strings, trumpets are brass instruments.
- **Events:** Events are the actions which occur in our world.
- **Performance:** It describes behavior which involves knowledge about how to do things.
- **Meta-knowledge:** It is knowledge about what we know.
- **Facts:** Facts are the truths about the real world and what we represent.
- **Knowledge-Base:** The central component of the knowledge-based agents is the knowledge base. It is represented as KB. The Knowledgebase is a group of the Sentences.
- **Knowledge:** Knowledge is awareness or familiarity gained by experiences of facts, data, and situations. Following are the types of knowledge in artificial intelligence:

3.1 Conceptual Dependency

Conceptual dependency theory was based on two assumptions:

- If two sentences have the same meaning, they should be represented the same, regardless of the particular words used.
- Information implicitly stated in the sentence should be represented explicitly.

- That is, any information which can be inferred from what is explicitly stated should be included in the representation. These assumptions have many implications for what a representation language should look like.
- The first assumption implies that representations must be general; that is, they must capture the similarities in meanings of synonyms.
- For example, since "get" and "receive" can be used synonymously in many contexts, their conceptual representations in these contexts ought to reflect this fact by consisting of similar, if not identical, predicates.
- The assumption that representations ought to explicitly represent implicit information means that inferences must be made in order to produce complete representations.
- If a sentence implies information without explicitly stating it, then in order to include that information in the representation, machinery must exist which can infer it from what is explicitly stated.
- This means that representations must support inference. In order to do this efficiently, they must be canonical. Whenever a particular inference can be made, it would be desirable if the same inference rule could always be used to make it.
- Without a canonical representation, several rules would be required, one for each different possible representation form.

Given these representational requirements, then, the vocabulary for conceptual dependency consisted of the following:

• A set of primitives, used to represent actions in the world

• A set of states, used to represent preconditions and results of actions

• A set of dependencies, or possible conceptual relationships which could exist between primitives, states, and the objects involved.

Three elemental kinds of concepts: a nominal and an action together with their modifiers.

➢

Concept —can be→ a nominal

- An abstract or concrete object that invokes an image;
- Cars are concrete objects;
- Gravity is an abstract concept;
- A nominal produces a picture (PP)

➢

Concept —can be→ an action

- What a nominal does?
- Something an animate nominal does to an object;
- There are primitive Actions and derived Actions;

➢

Concept —can be→ a modifier

- A modifier modifies a nominal or an action;
- A modifier specifies an action or a nominal;

3.2 Frames

- "Frame systems and semantic networks: These systems use the metaphor that objects are nodes in a graph that these nodes are organized in a taxonomic structure, and that links between nodes represent binary relations.
- In frame systems the binary relations are thought of as slots in one frame that are filled by another, whereas in semantic networks, they are thought of as arrows between nodes.
- The choice between the frame metaphor and the semantic network metaphor determines whether you draw the resulting networks as nested boxes or as

graphs, but the meaning and implementation of the two types of systems can be identical.

- "Semantic network" to mean "semantic network or frame system." Examples of frame systems: OWL, FRAIL, KODIAK. Examples of semantic networks: SNEPS, NETL, Conceptual Graphs."

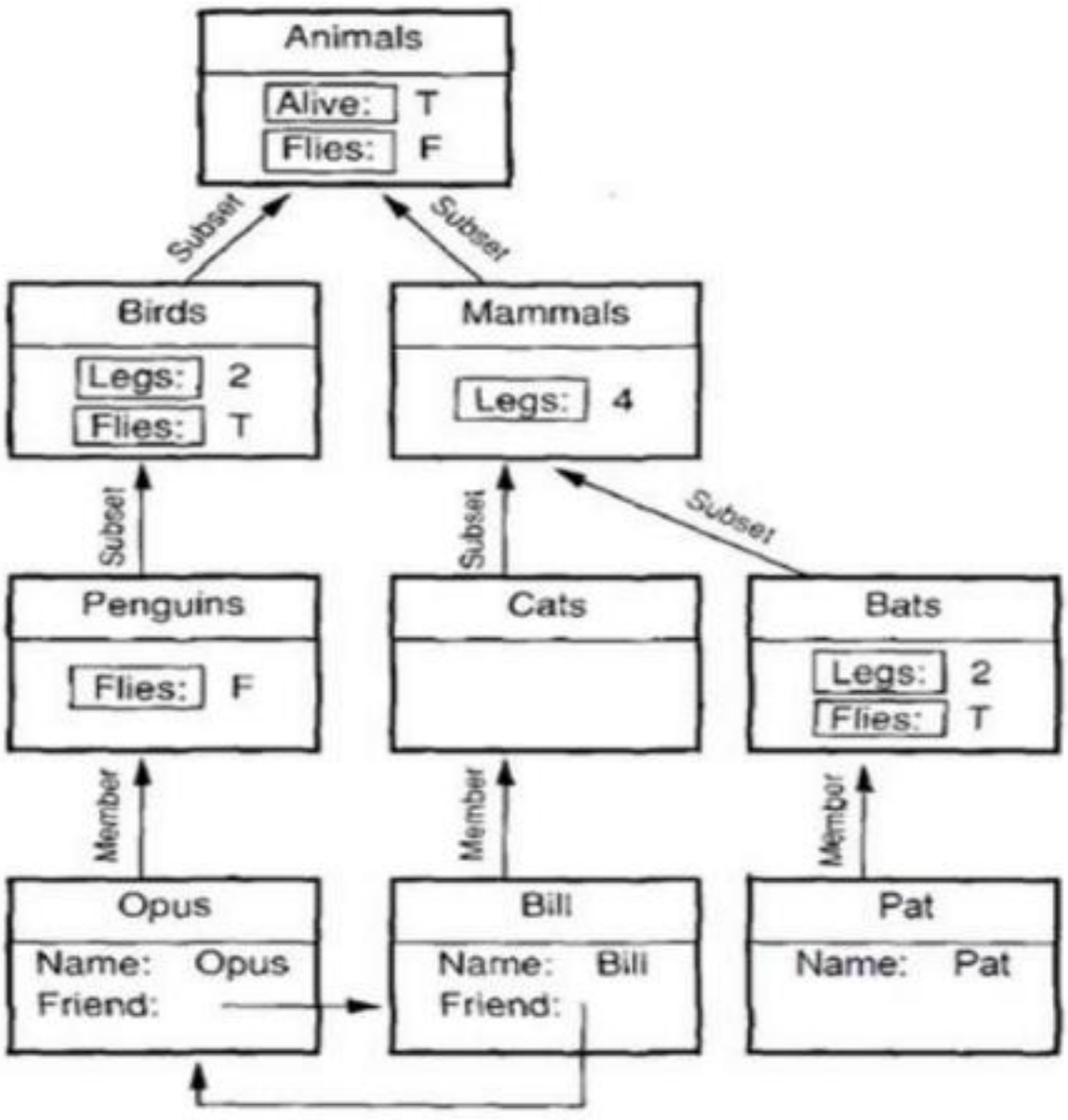

Fig – Frame based knowledge

- Natural language understanding requires inference i.e., assumptions about what is typically true of the objects or situations under consideration. Such information can be coded in structures known as frames.

NEED OF FRAMES

- Frame is a type of schema used in many AI applications including vision and natural language processing. Frames provide a convenient structure for representing objects that are typical to a stereotypical situation.
- The situations to represent may be visual scenes, structure of complex physical objects, etc. Frames are also useful for representing commonsense knowledge.

- As frames allow nodes to have structures they can be regarded as three-dimensional representations of knowledge.
- A frame is similar to a record structure and corresponding to the fields and values are slots and slot fillers. Basically it is a group of slots and fillers that defines a stereotypical object.
- A single frame is not much useful. Frame systems usually have collection of frames connected to each other. Value of an attribute of one frame may be another frame.

3.3 Semantic nets

- A semantic net is a knowledge representation technique used for propositional information. So it is also called a propositional net. Semantic nets convey meaning. They are two dimensional representations of knowledge. Mathematically a semantic net can be defined as a labelled directed graph.
- Semantic nets consist of nodes, links (edges) and link labels. In the semantic network diagram, nodes appear as circles or ellipses or rectangles to represent objects such as physical objects, concepts or situations.
- Links appear as arrows to express the relationships between objects, and link labels specify particular relations. Relationships provide the basic structure for organizing knowledge. The objects and relations involved need not be so concrete. As nodes are associated with other nodes semantic nets are also referred to as associative nets.

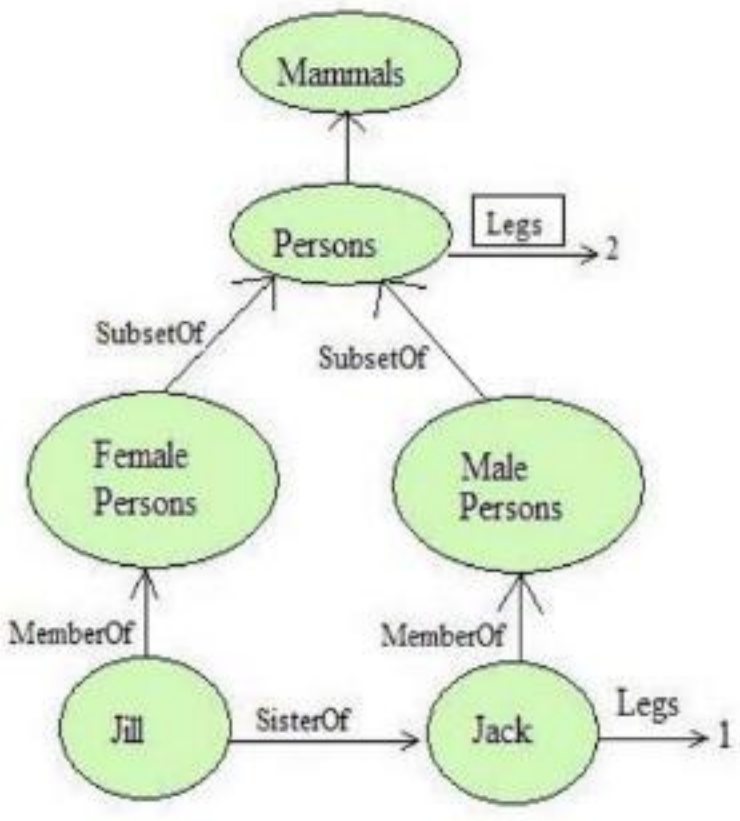

Fig - A Semantic Network

- In the above figure all the objects are within ovals and connected using labelled arcs. Note that there is a link between Jill and Female Persons with label Member Of.
- Similarly there is a Member Of link betwe en Jack and Mailpersons and Sister Of link betwe en Jill and Jack. The Member Of link between Jill and Female Persons indicates that Jill belongs to the category of female persons.
- A semantic network is a graphic notation for representing knowledge in patterns of interconnected nodes. Semantic networks became popular in artificial intelligence and natural language processing only because it represents knowledge or supports reasoning.
- These act as another alternative for predicate logic in a form of knowledge representation.
- The structural idea is that knowledge can be stored in the form of graphs, with nodes representing objects in the world, and arcs representing relationships between those objects.

Reference:

1. Nilsson Nils J , "Artificial Intelligence: A new Synthesis, Morgan Kaufmann Publishers Inc. San Francisco, CA, ISBN: 978-1-55-860467-4
2. Patrick Henry Winston, "Artificial Intelligence", Addison-Wesley Publishing Company, ISBN: 0-201-53377-4
3. Andries P. Engelbrecht-Computational Intelligence: An Introduction, 2nd Edition-Wiley India- ISBN: 978-0-470-51250-0
4. Deepak Khemani, "A First Course in Artificial Intelligence", McGraw Hill Education(India), 2013, ISBN : 978-1-25-902998-1
5. Elaine Rich, Kevin Knight and Nair, "Artificial Intelligence", TMH, ISBN-978-0-07-008770-5
6. Stuart Russell and Peter Norvig, "Artificial Intelligence: A Modern Approach", Third edition, Pearson, 2003, ISBN :10: 0136042597

Unit IV Natural Language Processing and ANN

1. Natural Language Processing:
   1.1 Introduction
   1.2 Stages in natural language Processing
   1.3 Application of NLP in Machine Translation
2. Learning:
   2.1 Supervised
   2.2 Unsupervised and Reinforcement learning
3. Artificial Neural Networks (ANNs):
   3.1 Concept
   3.2 Feed forward and Feedback ANNs
   3.3 Error Back Propagation
   3.4 Boltzmann Machine

1. Natural Language Processing:

- Natural Language Processing is the technology used to aid computers to understand the human's natural language.
- It's not an easy task teaching machine to understand how we communicate.
- Leand Romaf, an experienced software engineer who is passionate at teaching people how artificial intelligence systems work, says that "in recent years, there have been significant breakthroughs in empowering computers to understand language just as we do."
- This article will give a simple introduction to Natural Language Processing and how it can be achieved.

1.1 Introduction

- Natural Language Processing, usually shortened as NLP, is a branch of artificial intelligence that deals with the interaction between computers and humans using the natural language.
- The ultimate objective of NLP is to read, decipher, understand, and make sense of the human languages in a manner that is valuable.
- Most NLP techniques rely on machine learning to derive meaning from human languages.
- In this NLP AI Tutorial, we will study what is NLP in Artificial Language. Moreover, we will discuss the components of Natural Language Processing and NLP applications. Along with this, we will learn the process, steps, importance and examples of NLP.

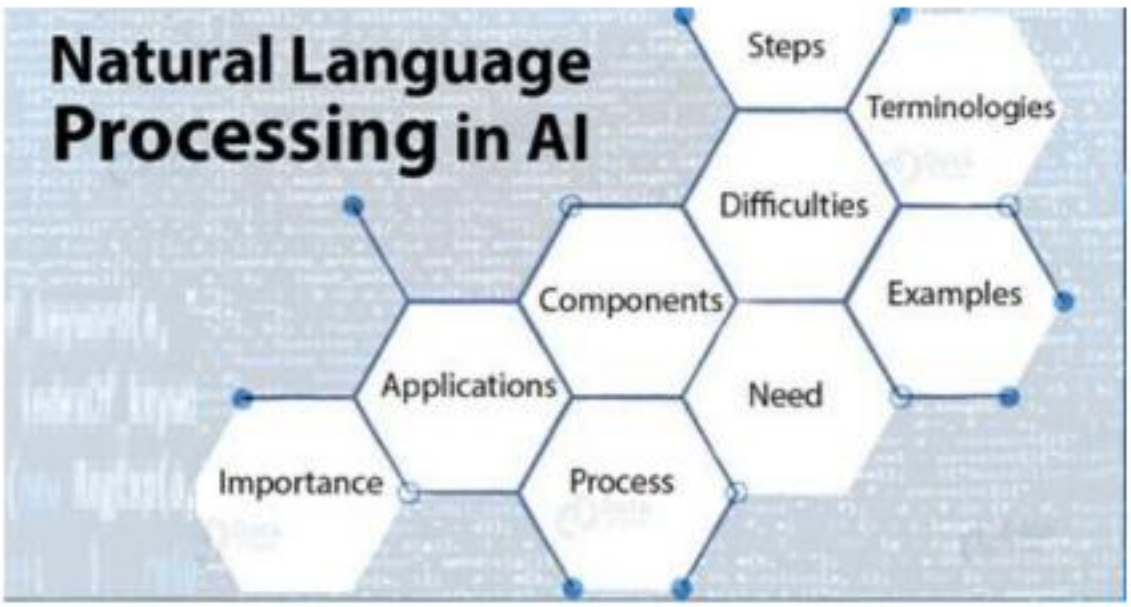

Fig - Natural Language Processing

We use the English language to communicate between an intelligent system and N.L.P. Processing of Natural Language plays an important role in various systems.

For Example:
A robot, it is used to perform as per your instructions. The input and output of an N.L.P system can be –

- Speech
- Written Text

- Natural Language processing is considered a difficult problem in computer science. It's the nature of the human language that makes NLP difficult.
- The rules that dictate the passing of information using natural languages are not easy for computers to understand.
- Some of these rules can be high-leveled and abstract; for example, when someone uses a sarcastic remark to pass information.
- On the other hand, some of these rules can be low-levelled; for example, using the character "s" to signify the plurality of items.

1.2 Stages in natural language Processing

- Natural Language Processing (NLP) refers to AI method of communicating with intelligent systems using a natural language such as English.
- Processing of Natural Language is required when you want an intelligent system like robot to perform as per your instructions, when you want to hear decision from a dialogue based clinical expert system, etc.
- The field of NLP involves making computers to perform useful tasks with the natural languages humans use.

Stages in NLP
There are general five stages –

- **Lexical Analysis** – It involves identifying and analyzing the structure of words. Lexicon of a language means the collection of words and phrases in a language. Lexical analysis is dividing the whole chunk of txt into paragraphs, sentences, and words.
- **Syntactic Analysis (Parsing)** – It involves analysis of words in the sentence for grammar and arranging words in a manner that shows the relationship among the words. The sentence such as "The school goes to boy" is rejected by English syntactic analyzer.

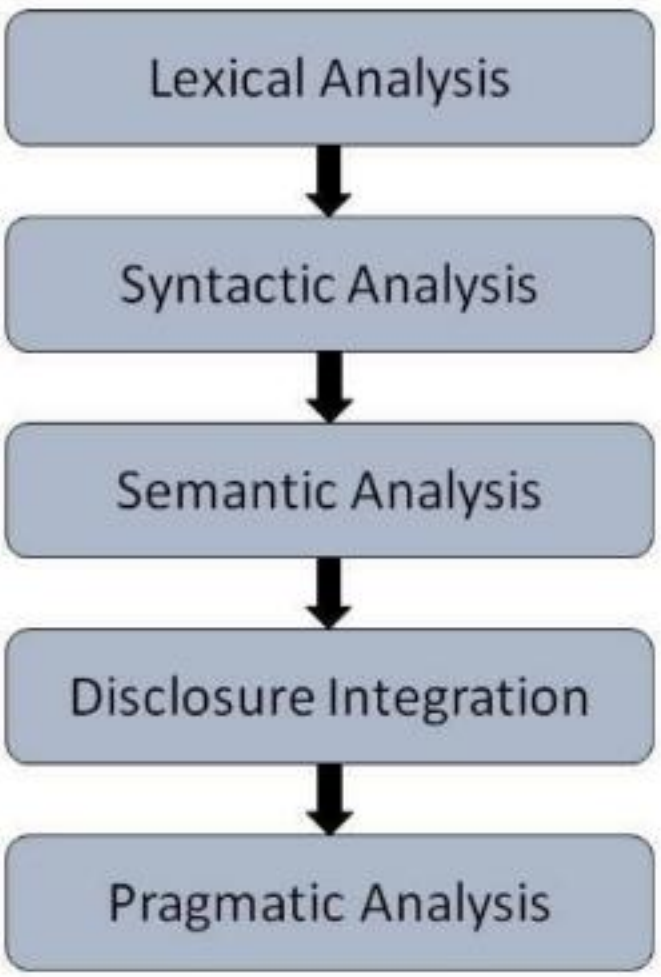

Fig - Stages in natural language Processing

- **Semantic Analysis** – It draws the exact meaning or the dictionary meaning from the text. The text is checked for meaningfulness. It is done by mapping syntactic structures and objects in the task domain. The semantic analyzer disregards sentence such as "hot ice-cream".
- **Discourse Integration** – the meaning of any sentence depends upon the meaning of the sentence just before it. In addition, it also brings about the meaning of immediately succeeding sentence.
- **Pragmatic Analysis** – during this, what was said is re-interpreted on what it actually meant. It involves deriving those aspects of language which require real world knowledge.

1.3 Application of NLP in Machine Translation

- Natural Language Processing is the driving force behind the following common applications:
- Language translation applications such as Google Translate
- Word Processors such as Microsoft Word and Grammar that employ NLP to check grammatical accuracy of texts.

- Interactive Voice Response (IVR) applications used in call centers to respond to certain users' requests.
- Personal assistant applications such as OK Google, Sire, Cortana, and Alexa.
- Machine translation (MT), process of translating one source language or text into another language, is one of the most important applications of NLP. We can understand the process of machine translation with the help of the following flowchart –

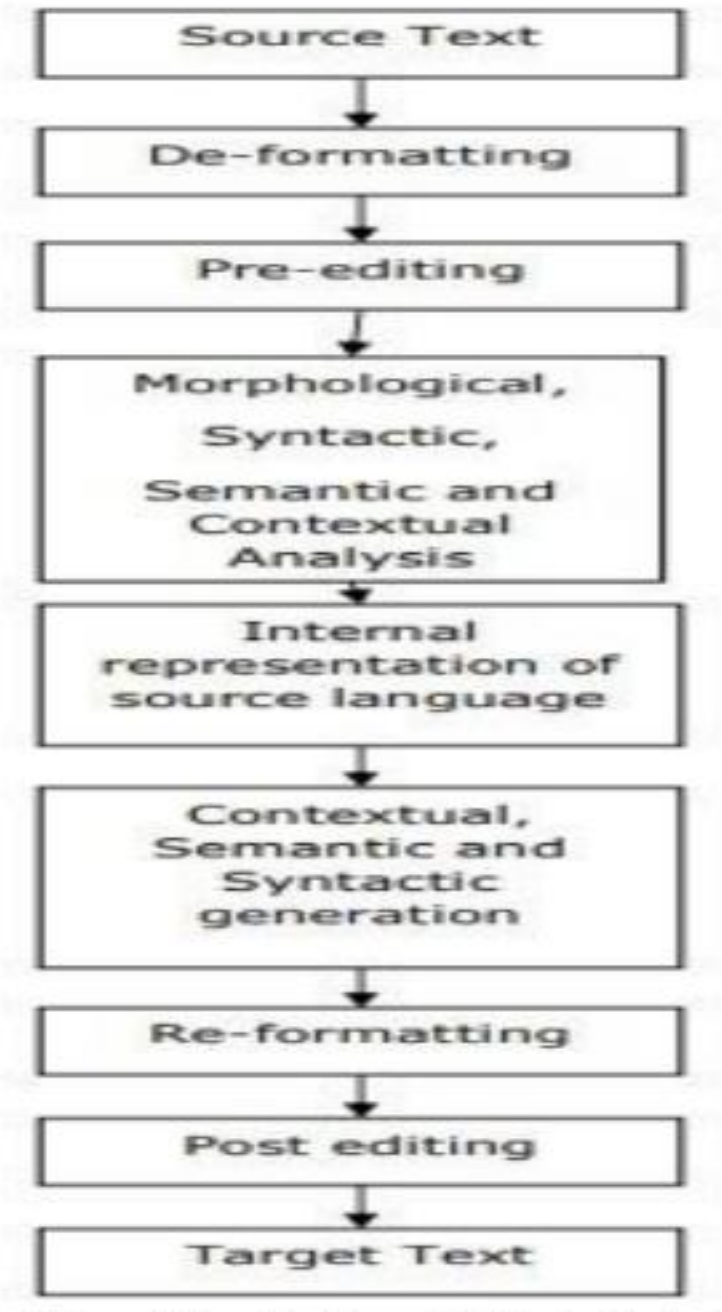

Fig – Representation of Application of NLP in Machine Translation

Types of Machine Translation Systems

There are different types of machine translation systems. Let us see what the different types are.

- **Bilingual MT System**
  Bilingual MT systems produce translations between two particular languages.

- **Multilingual MT System**
  Multilingual MT systems produce translations between any pair of languages. They may be either uni-directional or bi-directional in nature.
- **Approaches to Machine Translation (MT)**
  Let us now learn about the important approaches to Machine Translation. The

Approaches to MT are as follows –

- **Direct MT Approach**
  It is less popular but the oldest approach of MT. The systems that use this approach are capable of translating SL (source language) directly to TL (target language). Such systems are bi-lingual and uni-directional in nature.
- **Interlingua Approach**
  The systems that use Interlingua approach translate SL to an intermediate language called Interlingua (IL) and then translate IL to TL. The Interlingua approach can be understood with the help of the following MT pyramid –

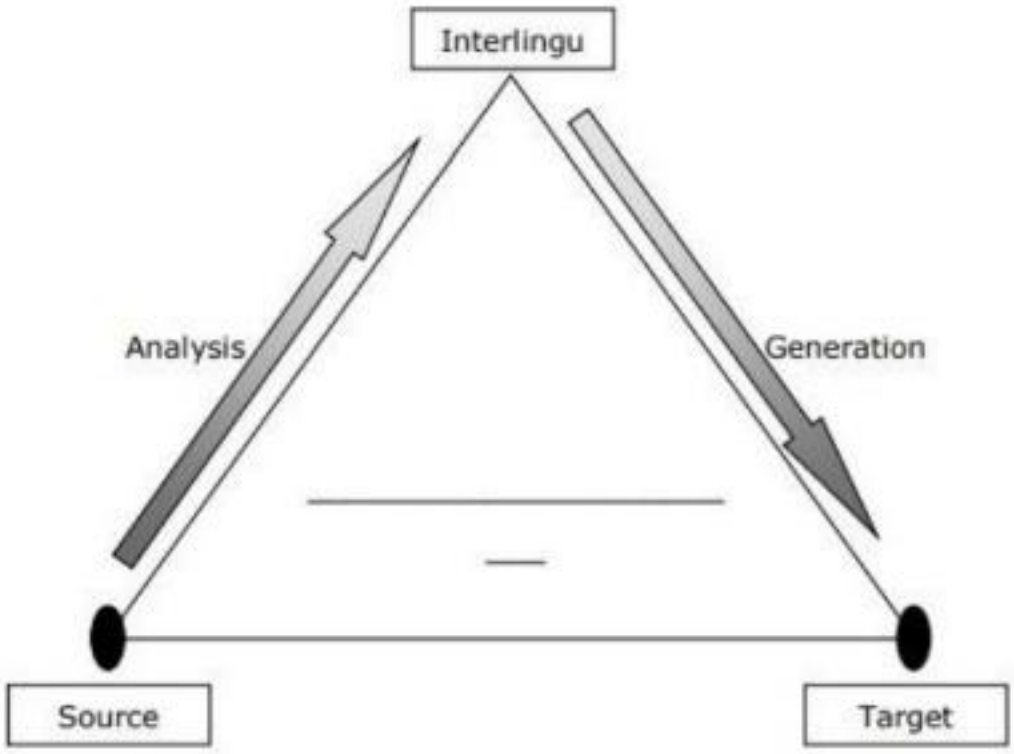

Fig - Approaches to MT

2. Learning:

2.1 Supervised

- It involves a teacher that is scholar than the ANN itself.
- For example, the teacher feeds some example data about which the teacher already knows the answers.
- For example, pattern recognizing. The ANN comes up with guesses while recognizing.

- Then the teacher provides the ANN with the answers. The network then compares it guesses with the teacher's "correct" answers and makes adjustments according to errors.

2.2 Unsupervised and Reinforcement learning

- Unsupervised Learning – It is required when there is no example data set with known answers.
- For example, searching for a hidden pattern.
- In this case, clustering i.e. dividing a set of elements into groups according to some unknown pattern is carried out based on the existing data sets present.
- Reinforcement Learning – this strategy built on observation.
- The ANN makes a decision by observing its environment.
- If the observation is negative, the network adjusts its weights to be able to make a different required decision the next time.

3. Artificial Neural Networks (ANNs):

- Neural networks are parallel computing devices, which is basically an attempt to make a computer model of the brain. The main objective is to develop a system to perform various computational tasks faster than the traditional systems.
- These tasks include pattern recognition and classification, approximation, optimization, and data clustering.
- Artificial Neural Network (ANN) is an efficient computing system whose central theme is borrowed from the analogy of biological neural networks. ANNs are also named as "artificial neural systems," or "parallel distributed processing systems," or "connectionist systems."
- ANN acquires a large collection of units that are interconnected in some pattern to allow communication between the units. These units, also referred to as nodes or neurons, are simple processors which operate in parallel.
- Every neuron is connected with other neuron through a connection link. Each connection link is associated with a weight that has information about the input signal. This is the most useful information for neurons to solve a particular problem because the weight usually excites or inhibits the signal that is being communicated.
- Each neuron has an internal state, which is called an activation signal. Output signals, which are produced after combining the input signals and activation rule, may be sent to other units.

3.1 Concept

The history of ANN can be divided into the following three eras –
ANN during 1940s to 1960s
Some key developments of this era are as follows –

- **1943** – It has been assumed that the concept of neural network started with the work of physiologist, Warren McCulloch, and mathematician, Walter Pitts, when in 1943 they modelled a simple neural network using electrical circuits in order to describe how neurons in the brain might work.
- **1949** – Donald Hebb's book, The Organization of Behavior, put forth the fact that repeated activation of one neuron by another increases its strength each time they are used.
- **1956** – an associative memory network was introduced by Taylor.
- **1958** – A learning method for McCulloch and Pitts neuron model named Perceptron was invented by Rosenblatt.
- **1960** – Bernard Widrow and Marcian Hoff developed models called "ADALINE" and "MADALINE."

ANN during 1960s to 1980s
Some key developments of this era are as follows –

- **1961** – Rosenblatt made an unsuccessful attempt but proposed the "back propagation" scheme for multilayer networks.
- **1964** – Taylor constructed a winner-take-all circuit with inhibitions among output units.
- **1969** – Multilayer perceptron (MLP) was invented by Minsky and Papert.
- **1971** – Kohonen developed Associative memories.
- **1976** – Stephen Grossberg and Gail Carpenter developed Adaptive resonance theory.

3.2 Feed forward and Feedback ANNs

- Artificial neural networks are described by three components. The first is the model's architecture, or topology, which describes the layers of neurons and structure of the connections between them
- The second component is the activation function used by the artificial neurons. The third component is the learning algorithm that finds the optimal values of the weights.
- There are two main types of artificial neural networks. Feedforward neural networks are the most common type of neural net, and are defined by their directed acyclic graphs. Signals only travel in one direction—towards the output layer—in feedforward neural networks.
- Conversely, feedback neural networks, or recurrent neural networks, do contain cycles. The feedback cycles can represent an internal state for the network that can cause the network's behavior to change over time based on its input.

Feedforward neural networks are commonly used to learn a function to map an input...

FeedForward ANN

- ANNs allow signals to travel one way only: from input to output. There is no feedback (loops); i.e., the output of any layer does not affect that same layer. Feed-forward ANNs tend to be straightforward networks that associate inputs with outputs. They are extensively used in pattern recognition. This type of organisation is also referred to as bottom-up or top-down.
- In this ANN, the information flow is unidirectional. A unit sends information to other unit from which it does not receive any information. There are no feedback loops. They are used in pattern generation / recognition/classification. They have fixed inputs and outputs.

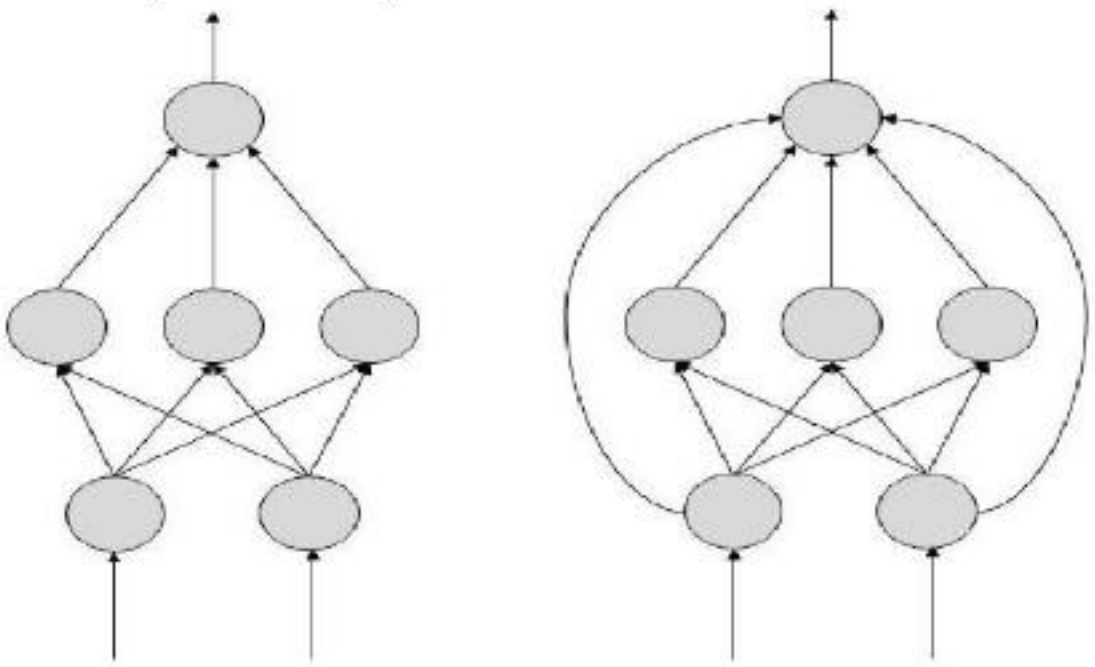

Fig – Representation of FeedForward ANN

FeedBack ANN:

- Networks can have signals travelling in both directions by introducing loops in the network. Feedback networks are powerful and can get extremely complicated. Computations derived from earlier input are fed back into the network, which gives them a kind of memory.
- Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found.
- Here, feedback loops are allowed. They are used in content addressable memories.

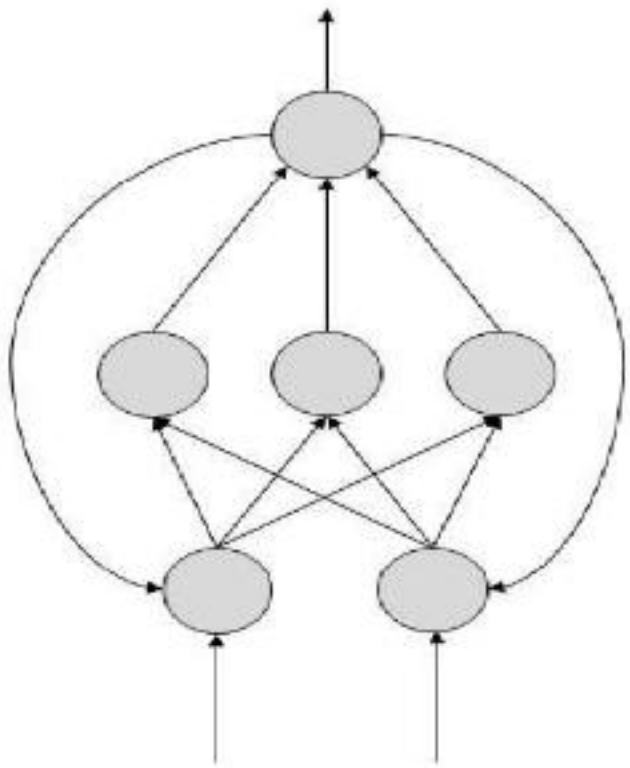

Fig – Representation of FeedBack ANN:

3.3 Error Back Propagation

- Backpropagation, short for "backward propagation of errors," is an algorithm for supervised learning of artificial neural networks using gradient descent. Given an artificial neural network and an error function the method calculates the gradient of the error function with respect to the neural network's weights. It is a generalization of the delta rule for perceptrons to multilayer feedforward neural networks.
- The "backwards" part of the name stems from the fact that calculation of the gradient proceeds backwards through the network, with the gradient of the final layer of weights being calculated first and the gradient of the first layer of weights being calculated last.
- Partial computations of the gradient from one layer are reused in the computation of the gradient for the previous layer. This backwards flow of the error information allows for efficient computation of the gradient at each layer versus the naive approach of calculating the gradient of each layer separately.
- Back propagation's popularity has experienced a recent resurgence given the widespread adoption of deep neural networks for image recognition and speech recognition. It is considered an efficient algorithm, and modern implementations take advantage of specialized GPUs to further improve performance.
- Basically, what we need to do, we need to somehow explain the model to change the parameters (weights), such that error becomes minimum.
  Let's put it in a way, we need to train our model.

- One way to train our model is called as Backpropagation. Consider the diagram below:

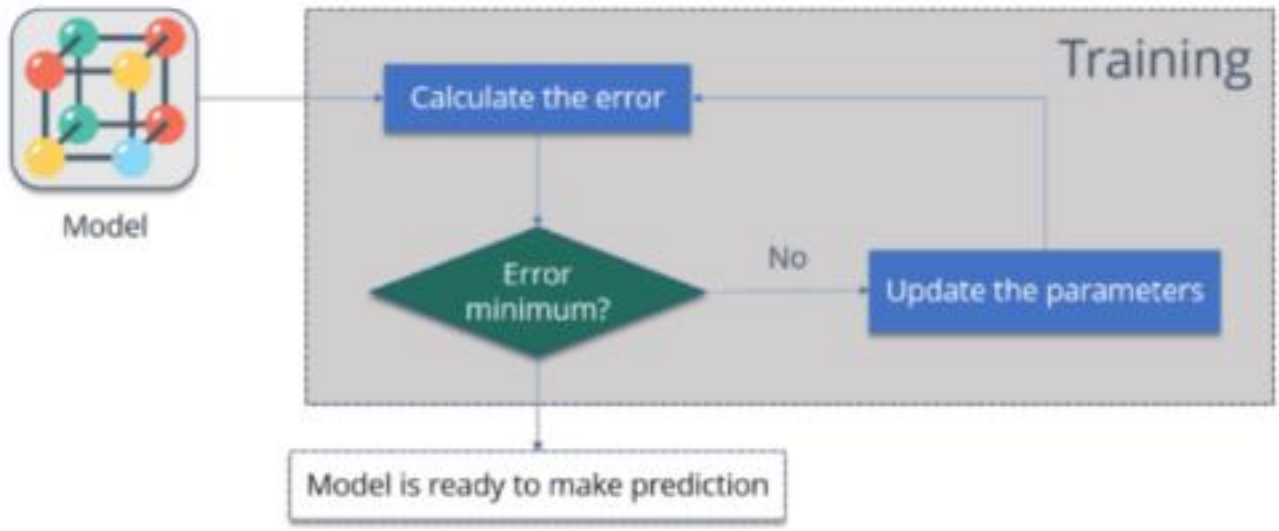

Fig - Error Back Propagation

Let me summarize the steps for you:

- **Calculate the error** – How far is your model output from the actual output.
- **Minimum Error** – Check whether the error is minimized or not.
- **Update the parameters** – If the error is huge then, update the parameters (weights and biases). After that again check the error. Repeat the process until the error becomes minimum.
- **Model is ready to make a prediction** – Once the error becomes minimum, you can feed some inputs to your model and it will produce the output.

3.4 Boltzmann Machine

- A Boltzmann machine is a type of stochastic recurrent neural network and Markov random field. Boltzmann machines can be seen as the stochastic, generative counterpart of Hopfield networks. They were one of the first neural networks capable of learning internal representations, and are able to represent and (given sufficient time) solve difficult combinatory problems.
- They are theoretically intriguing because of the locality and Hebbian nature of their training algorithm (being trained by Hebb's rule), and because of their parallelism and the resemblance of their dynamics to simple physical processes.
- Boltzmann machines with unconstrained connectivity have not proven useful for practical problems in machine learning or inference, but if the connectivity is properly constrained, the learning can be made efficient enough to be useful for practical problems.

- They are named after the Boltzmann distribution in statistical mechanics, which is used in their sampling function. That's why they are called "energy based models" (EBM). They were invented in 1985 by Geoffrey Hinton, then a Professor at Carnegie Mellon University, and Terry Sejnowski, then a Professor at Johns Hopkins University.
- A graphical representation of an example Boltzmann machine. Each undirected edge represents dependency. In this example there are 3 hidden units and 4 visible units. This is not a restricted Boltzmann machine.

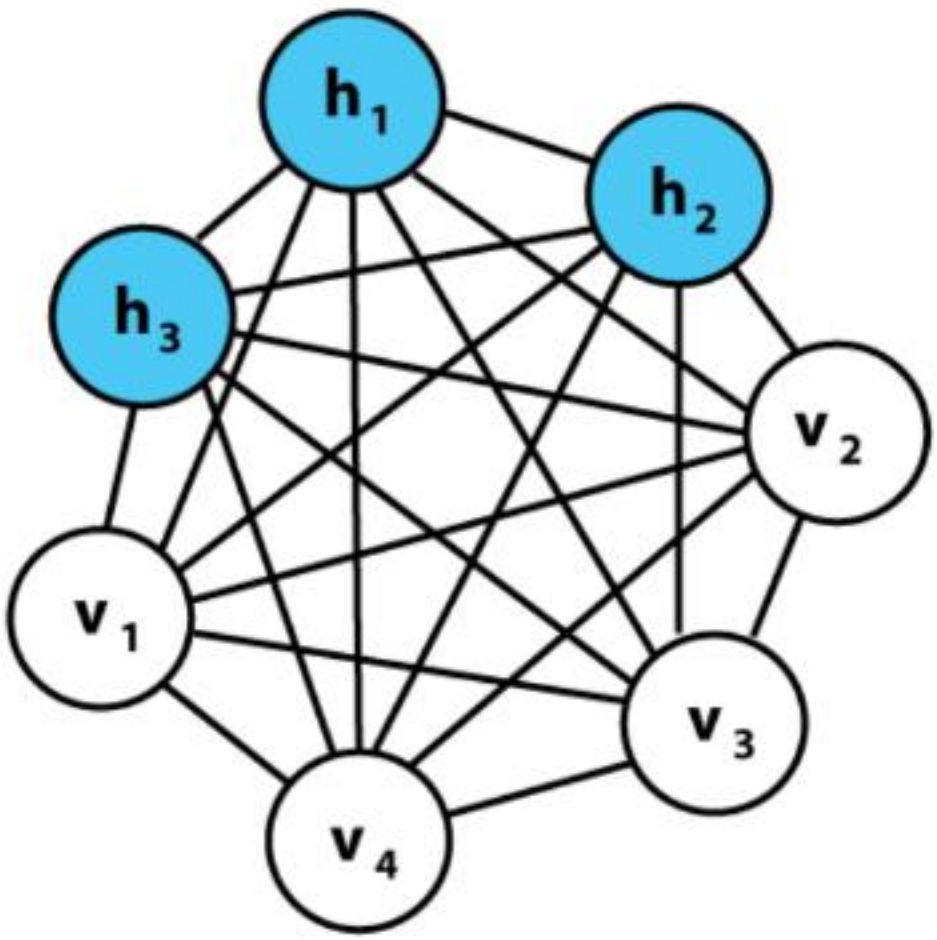

Fig - A graphical representation of an example Boltzmann machine

- Boltzmann machines are used to solve two quite different computational problems. For a search problem, the weights on the connections are fixed and are used to represent a cost function. The stochastic dynamics of a Boltzmann machine then allow it to sample binary state vectors that have low values of the cost function.
- For a learning problem, the Boltzmann machine is shown a set of binary data vectors and it must learn to generate these vectors with high probability. To do this, it must find weights on the connections so that, relative to other possible binary vectors, the data vectors have low values of the cost function.

- To solve a learning problem, Boltzmann machines make many small updates to their weights, and each update requires them to solve many different search problems.

Reference:

1. Stuart Russell and Peter Norvig, "Artificial Intelligence: A Modern Approach", Third edition, Pearson, 2003, ISBN :10: 0136042597
2. Michael Jenkin, Gregory, " Computational Principals of Mobile Robotics", Cambridge University Press, 2010, ISBN : 978-0-52-187157-0
3. Nilsson Nils J , "Artificial Intelligence: A new Synthesis, Morgan Kaufmann Publishers Inc. San Francisco, CA, ISBN: 978-1-55-860467-4
4. Patrick Henry Winston, "Artificial Intelligence", Addison-Wesley Publishing Company, ISBN: 0-201-53377-4
5. Andries P. Engelbrecht-Computational Intelligence: An Introduction, 2nd Edition-Wiley India- ISBN: 978-0-470-51250-0

Unit V Robotics

1. Robotics:
   1.1 Fundamentals
   1.2 Path Planning for Point Robot
   1.3 Mobile Robot Hardware
2. Non Visual Sensors like:
   2.1 Contact Sensors
   2.2 Inertial Sensors
   2.3 Infrared Sensors
   2.4 Sonar and Radar
   2.5 Laser Rangefinders
   2.6Biological Sensing
3. Robot System Control:
   3.1 Horizontal and Vertical Decomposition
   3.2 Hybrid Control Architectures
   3.3 Middleware
   3.4 High-Level Control
   3.5 Human-Robot Interface

1. Robotics:

- Robotics is a domain in artificial intelligence that deals with the study of creating intelligent and efficient robots. Robots are the artificial agents acting in real world environment.
- Robots are aimed at manipulating the objects by perceiving, picking, moving, modifying the physical properties of object, destroying it, or to have an effect thereby freeing manpower from doing repetitive functions without getting bored, distracted, or exhausted.
- Robotics is a branch of AI, which is composed of Electrical Engineering, Mechanical Engineering, and Computer Science for designing, construction, and application of robots.

Aspects of Robotics

- The robots have mechanical construction, form, or shape designed to accomplish a particular task.
- They have electrical components which power and control the machinery.
- They contain some level of computer program that determines what, when and how a robot does something.

1.1 Fundamentals

- Traditionally, this system aspect of the robot has been overlooked in robotics textbooks. Although an information system is not too important to an industrial robot, it is an indispensable subsystem of the humanoid robot.
- Here, we emphasize the basics of the information system, in terms of data processing, storage x The Fundamentals of Robotics: Linking Perception to Action and communication.
- We study the fundamentals of computing platforms, micro-controllers and programming together with a conceptual description of multi-tasking. I also include discussions on various interfacing systems typically used by a micro-controller (i.e. I/O).
- It is a common fallacy among novices that it is simple to form a vision system by putting optical lenses, cameras and computers together. However, the flow of information from the light rays to the digital images goes through various signal conversions.
- Thus, a primary concern of the robot's visualsensory system is whether or not the loss of information undermines the image and vision computing by the robot's visual-perception system. I start this chapter with a study of the basic properties of light and human vision.

- Then, I focus on the detailed working principles underlying optical imageformation, electronic image-formation, and the mathematical modelling of the robot's visual sensory system.

1.2 Path Planning for Point Robot

- Path-planning requires a map of the environment and the robot to be aware of its location with respect to the map. We will assume for now that the robot is able to localize itself, is equipped with a map, and capable of avoiding temporary obstacles on its way.
- How to create a map, how to localize a robot, and how to deal with uncertain position information will be major foci of the reminder of this class. The goals of this lecture are introduce suitable map representations explain basic path-planning algorithms ranging from Dijkstra, to A*, D* and RRT introduce variations of the path-planning problem, such as coverage path planning Map representations.
- In order to plan a path, we somehow need to represent the environment in the computer. We differentiate between two complementary approaches: discrete and continuous approximations. In a discrete approximation, a map is sub-divided into chunks of equal or differing sizes (e.g., rooms in a building). The latter maps are also known as topological maps.
- Discrete maps lend themselves well to a graph representation. Here, every chunk of the map corresponds to a vertex (also known as "node"), which are connected by edges, if a robot can navigate from one vertex to the other.
- For example a road-map is a topological map, with intersections as vertices and roads as edges. Computationally, a graph might be stored as an adjacency or incidence list/matrix. A continuous approximation requires the definition of inner (obstacles) and outer boundaries, typically in the form of a polygon, whereas paths can be encoded as sequences of real numbers.
- Discrete maps are the dominant representation in robotics. Currently the most common map is the occupancy grid map. In a grid map, the environment is discredited into squares of arbitrary resolution, e.g. 1cm x 1cm, on which obstacles is marked.
- In a probabilistic occupancy grid, grid cells can also be marked with the probability that they contain an obstacle. This is particularly important when the position of the robot that senses an obstacle is uncertain.
- Disadvantages of grid maps are their large memory requirements as well as computational time to traverse data structures with large numbers of vertices. A solution to the latter problem is topological maps that encode entire rooms as vertices and use edges to indicate navigable connections between them.
- There is no silver bullet, and each application might require a different solution that could be a combination of different map types.

- There exists also every possible combination of discrete and continuous representation. For example, roadmaps for GPS systems are stored as topological maps that store the GPS coordinates of every vertex.

- **Path-Planning Algorithms**

  The problem to find a "shortest" path from one vertex to another through a connected graph is of interest in multiple domains, most prominently in the internet, where it is used to find an optimal route for a data packet.
  The term "shortest" refers here to the minimum cumulative edge cost, which could be physical distance (in a robotic application), delay or any other metric that is important for a specific application.

Fig - Path Planning for Point Robot

- Animation of Dijkstra's algorithm for robotic path planning Subhrajit Bhattachary One of the earliest and simplest algorithms is Dijkastra's algorithm. Starting from the initial vertex where the path should start, the algorithm marks all direct neighbors of the initial vertex with the cost to get there.
- It then proceeds from the vertex with the lowest cost to all of its adjacent vertices and marks them with the cost to get to them via itself if this cost is lower. Once all neighbors of a vertex have been checked, the algorithm proceeds to the vertex with the next lowest cost.
- A more detailed description of this algorithm is provided here. Once the algorithm reaches the goal vertex, it terminates and the robot can follow the edges pointing towards the lowest edge cost.

1.3 Mobile Robot Hardware

It is interesting to remark that any hardware and software architecture should be particularized for a given mobile robot and different solutions can be efficiently implemented on different platforms. The following evaluation criteria can be applied for selecting the most appropriate architecture:

- Modularity.
- Ease of implementation.
- Platform portability.
- Robustness.
- Solution cost.
- We firstly propose hardware and software architecture for an industrial forklift to be used in the automation of transport processes. A proposed architecture for a convoying application with an electric car is secondly showed.
- Then, a RT platform is proposed for the commercial robot Robuter n, and architecture is described for a small differential robot, named PCBot n, both used for research applications.
- The extensible hardware architecture described is currently implemented on the K9 rover and is being transplanted to two K10 robots with the goal of supporting research into peer-to-peer human-robot collaboration.
- These two new robots vary from K9 both in instrumentation and capabilities. However, we aim to show that K9's extensible architecture is scalable enough to meet the needs of K10 with minimal redesign.

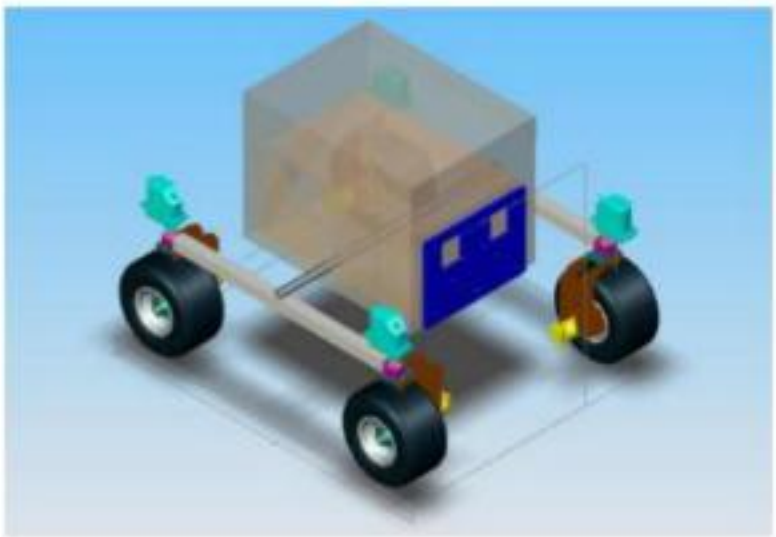

Fig – Architecture of Mobile Robot Hardware

- Each robot features a 4-wheel steer, 4-wheel drive rocker chassis and a top speed of 30 cm/sec, comparable to human-walking speed. Hard points on all sides allow attachment of additional components including antennas, masts, arms, and other equipment. K10's avionics strives to duplicate those of K9 wherever possible, including the computing infrastructure, power system, and instrumentation.
- Although the K10 robots will draw significantly more power than K9 (approx. 500W versus 100W nominal), K9's power architecture is extensible enough to accommodate K10's power requirements with minimal modification, thereby saving engineering time and development costs.

2. Non Visual Sensors like:

- Locate and read a recent full-text article from the Capella Library about one of the non-visual senses and a related phenomenon. Describe the phenomenon , the related sensory and brain structures, and how damage to one or more of these structures can impact the sensation or perception of this phenomenon.
- Respond to at least one other learner. Your response is expected to be substantive in nature and to reference the assigned readings, as well as other theoretical, empirical, or professional literature to support your views and writings. Reference your sources using standard APAguidelines.The ability to detect audition is considered by many as one of the most important senses, second only to vision.
- The receptive organ for auditory functioning is the Corti consisting of the basal membrane, hair cells, and tectorial membrane. The cochlea is part of the inner ear just inside middle ear canal, which responds to pressures changes within its fluid filled basal membrane.
- The cochlea is divided into three sections: the scala vestibule, scala media, and scala timpani. When sound is detected, the basal membrane moves laterally in conjunction with the tectorialmembrane which projects overhead.
- These movements of the fluid within the cochlea cause the cilia of the inner hair cells to vibrate, thus producing receptor potentials.
- These inner hair cells form synapses with the dendrites of the bipolar neurons which connect to the cochlear branch of the eighth cranial nerve (cochlear nerve)to send auditory messages to the brain.
- Although inner hair cells only form 29 percent of the total number of receptor cells, their connections are of vital importance to the transmission of auditory information to the central nervous system (CNS). This auditory information is relayed specifically to the primary auditory cortex on the medial surface of the temporal lobe.

2.1 Contact Sensors

- Contact sensors are those which require physical contact against other objects to trigger. A push button switch, limit switch or tactile bumper switch are all examples of contact sensors.

Fig - Contact Sensors

- These sensors are mostly used for obstacle avoidance robots. When these switches hit an obstacle, it triggers the robot to do a task, which can be reversing, turning, switching on a LED, Stopping etc. There are also capacitive contact sensors which react only to human touch (Not sure if they react to animals touch).
- Touch screen Smart phones available these days use capacitive touch sensors (Not to be confused with older stylus based models). Contact Sensors can be easily implemented, but the drawback is that they require physical contact.
- In other words, your robot will not turn until it hits an object. A better alternative is to use a proximity sensor.

2.2 Inertial Sensors

- Inertial sensors are sensors based on inertia. These range from MEMS inertial sensors, measuring only a few square mm, up to ring laser gyroscopes which are extremely accurate but can measure 50 cm in diameter.

➢ **Accelerometers – inertial sensors**

- MEMS inertial accelerometers consist of a mass-spring system, which reside in a vacuum. Exerting acceleration on the accelerometer results in a displacement of the mass in the spring system.
- The displacement of the mass depends on the mass-spring system, so a calibration is needed. Read-out can be via a capacitive system. MEMS accelerometers are available in 1D, 2D and 3D versions.
- As the size of the mass-spring system directly relates to the resolution (and accuracy) of the accelerometer, 3D accelerometer chip are not yet accurate enough to be used in high-accuracy MEMS AHRS's, such as the MTi-10, the MTi-100 and MTi-G-710. Main manufacturers of MEMS accelerometers are Analog Devices, Kionix and Colibrys.

➢ **Gyroscopes – inertial sensors**

- Inertial gyroscopes can be found in various classes. Ring Laser Gyroscopes (RLG) and Fiber Optic Gyros (FOG) are very reliable, and very expensive.
- They rely on light to be sent through a set of mirrors or a fiberglass cable in opposite direction. A rotation of the gyroscope results in light in one direction to reach the other side of the set of mirrors / fiberglass cable earlier than light sent away in the opposite direction.
- Optical gyroscopes, such as the RLG and FOG are so accurate; that they can be used without reference sensors (see AHRS). This intrinsic accuracy makes them so expensive that they cannot be used in cost-sensitive applications.

2.3 Infrared Sensors

- An infrared sensor is an electronic device that emits in order to sense some aspects of the surroundings. An IR sensor can measure the heat of an object as well as detects the motion.
- These types of sensors measures only infrared radiation, rather than emitting it that is called as a passive IR sensor. Usually in the infrared spectrum, all the objects radiate some form of thermal radiations.
- These types of radiations are invisible to our eyes that can be detected by an infrared sensor. The emitter is simply an IR LED (Light Emitting Diode) and the detector is simply an IR photodiode which is sensitive to IR light of the same wavelength as that emitted by the IR LED.
- When IR light falls on the photodiode, the resistances and these output voltages, change in proportion to the magnitude of the IR light received.

Fig - IR Sensor

IR Sensor Circuit Diagram and Working Principle

- An infrared sensor circuit is one of the basic and popular sensor modules in an electronic device. This sensor is analogous to human's visionary senses, which can be used to detect obstacles and it is one of the common applications in real time. This circuit comprises of the following components.

- LM358 IC 2 IR transmitter and receiver pair
- Resistors of the range of kilo ohms.
- Variable resistors.
- LED (Light Emitting Diode).

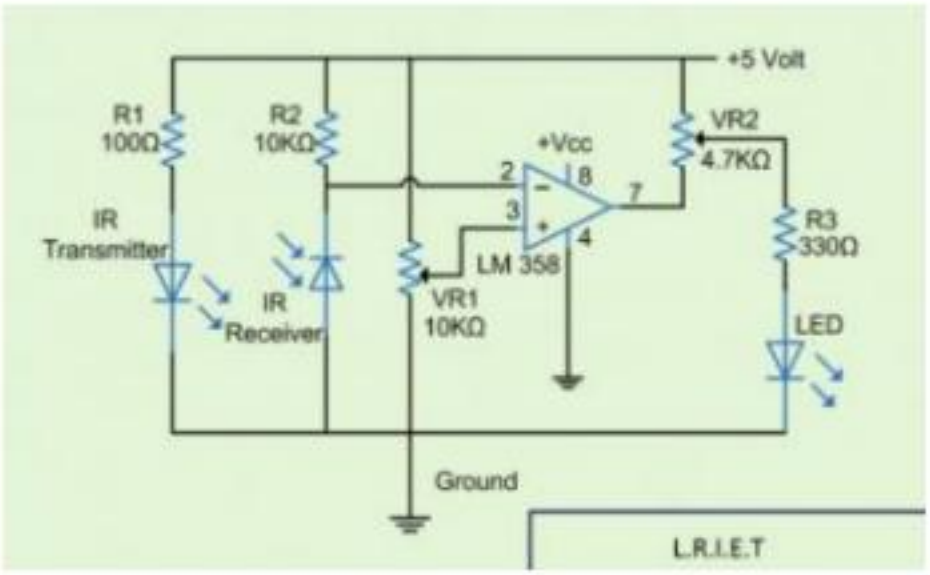

Fig - IR Sensor Circuit

- IR sensor, which transmits continuous IR, rays to be received by an IR receiver module. An IR output terminal of the receiver varies depending upon its receiving of IR rays. Since this variation cannot be analyzed as such, therefore this output can be fed to a comparator circuit. Here an operational amplifier (op-amp) of LM 339 is used as comparator circuit.

2.4 Sonar and Radar

- The word "sonar" comes from the first letters of "sound navigation ranging." Sonar can detect and locate objects under the sea by echoes, much as porpoises and other marine animals navigate using their natural sonar systems.
- There are two types of sonar sets: active and passive. An active sonar set sends out sound pulses called pings, and then receives the returning sound echo. Passive sonar sets receive sound echoes without transmitting their own sound signals.

- In active sonar sets, the sound signals are very powerful compared with ordinary sounds. Most sonar sets send out sounds that are millions of times more powerful than a shout. Each ping lasts a fraction of a second.
- Some sonar sets emit sounds you can hear. Other sonar signals are pitched so high that the human ear cannot hear them. These signals are called ultrasonic waves. The sonar set has a special receiver that can pick up the returning echoes. The location of underwater objects can then be determined by the length of time that elapses between sending the signal and hearing the returning echo.

Uses of Sonar

- Sonar has many uses. Submarines use sonar to detect other vessels. Sonar is also used to measure the depth of water, by means of a device called a Fathometer. (One fathom equals 6 feet, or about 1.8 meters.)
- The Fathometer measures the time it takes for a sound pulse to reach the bottom of the sea and return to the ship. Fishing boats use Fathometers to locate schools of fish.
- Oceanographers use sonar to map the contours of the ocean floor. Sound signals can also be sent into the mud or sand on the ocean floor and strike a layer of rock underneath. An echo then comes back, giving the distance to the rock layer.
- The same principle is used in searching for oil on land. A sonar pulse is sent into the ground. Echoes come back from the different layers of soil and rock and tell geologists what kinds of soils and rocks are present. This helps them identify areas for drilling that are most likely to contain oil or gas. This subterranean mapping is called seismic exploration.

Radar

- Radar sets, also called radar systems, come in many different sizes, depending on the job they are expected to do. But all have four main parts -- a transmitter, an antenna, a receiver, and an indicator (display screen).
- The transmitter produces the radio waves. When a radio wave strikes an object such as an airplane, part of the wave is reflected back to the radar set. The signal is detected by the antenna as a radio echo.
- The returning echo is sent to the receiver, where its strength is increased, or amplified. The echo is usually displayed as an image that can be seen on the indicator.
- The usual type of indicator is the plan position indicator, or PPI. On the face of its large tube, the operator sees a map-like picture of the surrounding region. This picture looks as if it were made looking down at the area from high above the radar set.

- On the indicator, the echoes appear as bright spots, called blips. The blips show where land areas are located. Blips also show the position of targets, such as planes and ships. The radar operator can pick out these targets because they are moving, while the land areas are not.
- A common type of radar is called pulse radar. This type of radar sends out radio waves in short bursts, or pulses. The distance to a target is determined by the time it takes the signal to reach the target and the echo to return.
- Radio signals travel at a known speed -- about 186,000 miles (300,000 kilometres) per second, the speed of light. If the radio signal comes back in 1/1,000 second, then the round trip is 186 miles (300 kilometres). The target must be half that far, or 93 miles (150 kilometres) away.
- Pulsed transmission helps determine the distance more accurately. Imagine that you are about to shout across a canyon to make an echo. If you shout a long sentence, the first words will come back before you can finish.
- It would be impossible to hear the entire echo clearly because it would be mixed with your own speech. But if you shout a short word the echo comes back crisp and clear with no interference from the transmitter (you).
- The location of the target in relation to the radar set is found in a different way. The radar antenna sends out radio pulses in a narrow beam, much like the beam of a flashlight. The antenna and its beam are rotated slowly through all possible directions, searching the entire horizon for targets.
- An echo is reflected from a ship or other object only if the narrow beam happens to strike it. The returning echoes are amplified by the receiver, then go to the indicator, which displays the range and direction of the target.

Uses of Radar

- Radar has both military and civilian uses. The most common civilian use is to help navigate ships and planes. Radar sets carried on a ship or located at an airport pick up echoes from other ships and planes and help prevent collisions.
- On ships, they also pick up echoes from buoys in channels when the ships enter or leave port. Radar sets help commercial airplanes land when visibility is bad or in the event of mechanical failure.
- Radar is also used in meteorology, including weather prediction. Weather forecasters use it, normally combined with lidar (optical radar), to study storms and locate hurricanes and blizzards.
- Doppler radar is based on the principle of the Doppler effect -- that is, the frequency of a wave changes as the source of the wave moves toward or away from the receiver. By analyzing changes in the frequency of reflected radio waves, Doppler radar can track the movement of storms and the development of tornadoes.

- An improved Doppler radar system called Next-Generation Radar (NEXRAD) can predict weather more accurately and farther into the future.
- Scientists use radar to track the migrations of birds and insects and to map distant planets. Because it can tell how fast and in which direction a target is moving, radar is used by police to locate speeding automobiles and control street traffic.
- Similar systems are used in tennis to measure the speed of serves and to call faults. Balloon-borne radar supports officials fighting drug trafficking. Surface-wave radar detects surface waves of the ocean to warn ships of icebergs and nearby vessels.

2.5 Laser Rangefinders

- A laser rangefinder is a rangefinder that uses a laser beam to determine the distance to an object. The most common form of laser rangefinder operates on the time of flight principle by sending a laser pulse in a narrow beam towards the object and measuring the time taken by the pulse to be reflected off the target and returned to the sender.
- Due to the high speed of light, this technique is not appropriate for high precision sub-millimetre measurements, where triangulation and other techniques are often used.
- Many laser rangefinders brands are now pay much attention on producing high accuracy rangefinders to meet customers' needs, functions of each also diverse a lot.
- Some are capable of offering slope distance, some are suitable for shaky hands, some are pretty suitable for long distance ranging, while some, ideal for measuring distances among woods and forests.
- Getting a best laser rangefinder is actually saving much money, its long service life and brilliant performance are able to provide you with satisfying using experience.
- How to avoid getting an inferior rangefinder? In this article, we will give you detailed best laser rangefinder reviews and buying guides on choosing them.

2.6 Biological Sensing

- Our world class capability in chemical and biological sensing technology is derived from a broad background in the physical mechanisms that lead to uniquely exploitable signatures, a strong knowledge of lasers, detectors, and optics that enables the capture of those signatures, as well as a foundation in applied mathematics and signal processing that enables the extraction of those signatures in highly cluttered environments.

- PSI's "laboratory to the field" approach to sensor development has led to the introduction of a range of sensor technologies. They include the Adaptive Infrared Imaging Spectroradiometer (AIRIS) with application to a range of standoff vapour, aerosol, and surface chemical and biological agent detection problems.
- Algorithm capability and applications knowledge has led to the development of active systems for the detection of surface chemical detection employing quantum cascade lasers as well as biological aerosol trigger detectors based on spark induced breakdown spectroscopy.
- PSI's C/B detection modalities are now being adapted to deployment on our Instant Eye UAV platform for sensitive site exploitation and combat outpost protection

3. Robot System Control:

- Robot arm control is a mechanical system control problem rather than a problem of controlling single actuators of a robot arm. Using principles and techniques of differential geometric system and control theory, a new dynamic system feedback technique is presented referenced to task space commands.
- In this task-driven dynamic control scheme the nonlinear robot arm system is feedback linear zed and simultaneously output decoupled by an appropriate nonlinear feedback and a nonlinear coordinate transformation.
- The new dynamic control technique actually transforms robot arm control problems from the joint space to the task space and performs robot servoing in terms of task space variables within a linear system frame, allowing also the use of powerful techniques of optimal control of linear systems.
- On the joint space level, the new dynamic control scheme only commands drive forces or torques or their equivalent quantities addressed to the joint drives.
- An important property of the new dynamic control technique is that the planned and commanded task space trajectory together with its time derivatives directly drives the robot arm through a linear system model.
- A method is briefly presented for task space motion planning matching the requirements of the new dynamic control scheme. It requires that the planned motion be presented to the controller as an input in form of a closed function of time.
- The implications of the new dynamic control technique are discussed for second and third order model robot arms with and without force feedback

measurements and for two (or more) dynamically cooperating robot arms using distinguished and in distinguished modeling for multiple cooperating arms.

3.1 Horizontal and Vertical Decomposition

- Objects responsible for too many things are a problem. Because their complexity is high, they are difficult to maintain and extend. Decomposition of responsibility is what we do in order to break these overly complex objects into smaller ones. I see two types of this refactoring operation: vertical and horizontal.
- Broadly speaking, AI companies can be categorised into Vertical AI and Horizontal AI. While Horizontal AI focuses on solving a larger problem statement, vertical AI focuses on niche.
- For instance, Apple's Siri or Amazon's Alexa are examples of Horizontal AI applications. As long as demand and resources exist, horizontal AI can develop AI technologies in-house.
- When they can't and the problem statement is too niche, they are generally acquired from vertical AI players in the market. Horizontal AI therefore focuses on more utility-like approaches for catering to the broad range of topics their end users are likely to need.
- Example, voice recognition has become mainstream AI nowadays and most of the large tech companies are well equipped with this technology. However an autonomous truck that offers last mile logistics services for warehouses based on deep learning insights from customer data is clearly a horizontal AI.
- The game therefore is simple, at some point there will be an industry dominated by incumbents (as we call it in traditional markets) bolting-on niche tech to complete itself. Horizontal AI will therefore combine with and enable Vertical AI.
- In the world of AI, these incumbents are often classified as **MAFIA** comprising of **M**icrosoft, **A**mazon, **F**ace book, **I**BM and **A**lphabet. Others include Apple Twitter, Intel, Baidu, and Alibaba.

3.2 Hybrid Control Architectures

- Three layer architectures seem to be the current state of evolution. Three layer architectures employ three levels of abstraction: the deliberative layer, a sequencing/executive layer and a reactive layer.
- Activities on the deliberative layer correspond to long term strategic planning as well as eventual plan adaptations. This level relies on very abstracted knowledge, highly sophisticated reasoning techniques and is the typical domain of AI planners able to make use of extensive application domain knowledge.

- The sequencing layer involves a reactive planner which selects and executes appropriate tactics using context dependent rules. A tactic is a pre-written ordered set of actions, rich in structure, i.e. disjunction and conjunction, recursion and hierarchies of actions. Execution of tactics finally leads to activation and termination of reactive layer behaviours.
- The reactive level performs the transition from symbolic reasoning to non-symbolic numerical control and the combination of separate behaviours.
- Within three layer architectures, communication and interaction is of greatest concern. The general approaches are:

- **Deliberative layer — sequencing layer**: The connection is done by mapping tactics to operator descriptions. Sequencing layer tactics are represented by planning operators in the deliberative layer. Parameters are bound by unification.
- **Sequencing layer — reactive layer**: Interaction between these two layers involve

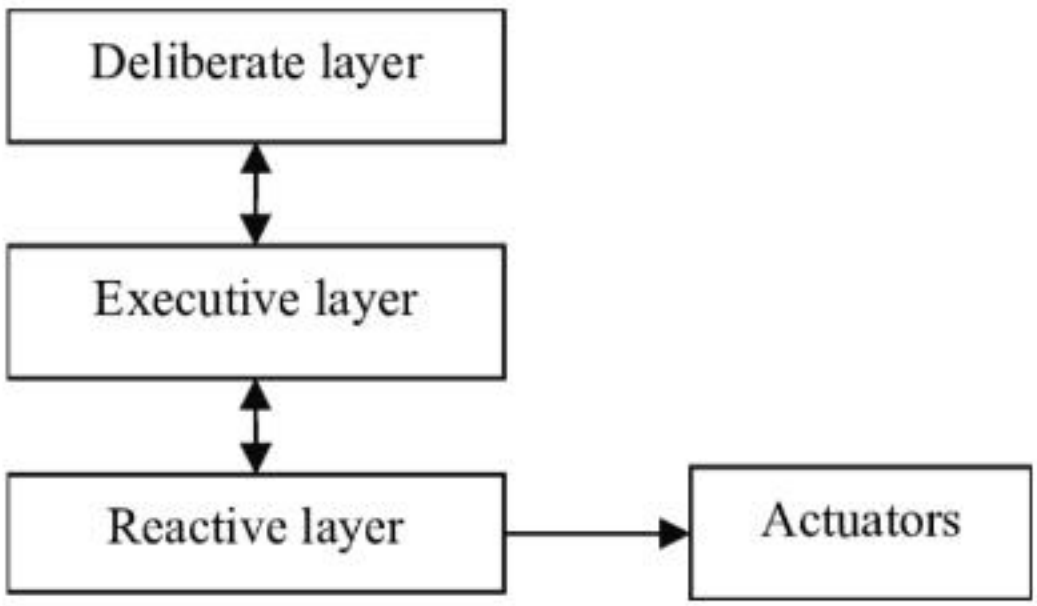

Fig - Hybrid Control Architectures

- activation;
- deactivation or termination;
- assign of control parameters; and
- Monitoring of success, as well as conversion of non-symbolic to symbolic parameters and vice-versa.

- The LAAS architecture is a representative instance of hybrid control systems based on several paradigms. The different agents to sense (perception modules), model (mapping modules), plan (planners) and act (actuator modules) indicate the observance of the SMPA paradigm.

- The hierarchy of levels implements a vertical decomposition. The modules, under certain circumstances, can operate in parallel to directly access predefined sensors and actuators, typifying a lateral decomposition of reactive sub-systems.
- An advantage of hybrid architectures is that, according to the considered application, some levels of the hierarchy can be suppressed. It can see an implementation without the higher planning level, unnecessary because of the small duration of the task.
- This example demonstrates the communication and interaction protocols between the modules, an executive and the decisional level in a real world application.

3.3 Middleware

- The reality is that not only is "artificial intelligence" a disparit term, encompasing a broad range of specifics, but "middleware" is a similarly generalized concept representing a variety of fruits from apples to oranges.
- Under the general rubric of "artificial intelligence middleware", I've identified four main categories: Multi-Agent System middleware, Game AI middleware, Dialog System middleware, and Robotics middleware.
- According to Wikipedia: "A multi-agent system (MAS) is a system composed of multiple interacting intelligent agents. Multi-agent systems can be used to solve problems that are difficult or impossible for an individual agent or a monolithic system to solve." According to Wikipedia: "Robotics middleware is middleware to be used in complex robot control software systems.
- It is the glue software that every system developer wants to use to connect software (and hardware) components together. Often, only communication between components is considered to be middleware".
- Dialog systems often involve the integration of multiple agents, and robotic systems may involve the integration of dialog systems. Game AI may include both "virtual robotics" and a dialog system layer.
- However, in dialog systems the middleware layer often seems to blur. Dialog systems themselves may perform a middleware function. The dialog system "dialog manager" component may serve middleware functions. And, additional middleware may interface with a dialog manager and/or dialog system.
- In fact, middleware and frameworks are often referred to in the same breath, as "middleware framework". To clarify, according to Wikipedia, middleware "software consists of a set of services that allows multiple processes running on one or more machines to interact"; whereas, a framework "is a collection of software libraries providing a defined application programming interface (API).

3.4 High-Level Control

- High-level control of the robot is the responsibility of an off-board computer that communicates with the robot via an RS232 port. The high-level controller can initiate low-level routines that are executed on-board, such as wall following, corner turning, and so on.
- Upon termination, these low-level routines communicate the status of the robot's sensors back to the high-level controller, which then decides how to proceed.
- The implementation details of the low-level actions are outside the scope of this paper, whose aim is to present the high-level controller.
- It should be noted that the aim of the paper is not to present advances in any particular sub-area of AI, such as planning, knowledge representation, or robotics, but rather to show how techniques from these areas can be notational medium and theorem proving as a means of cosynthesised and integrated into agent architecture, using logic as a represemputation.

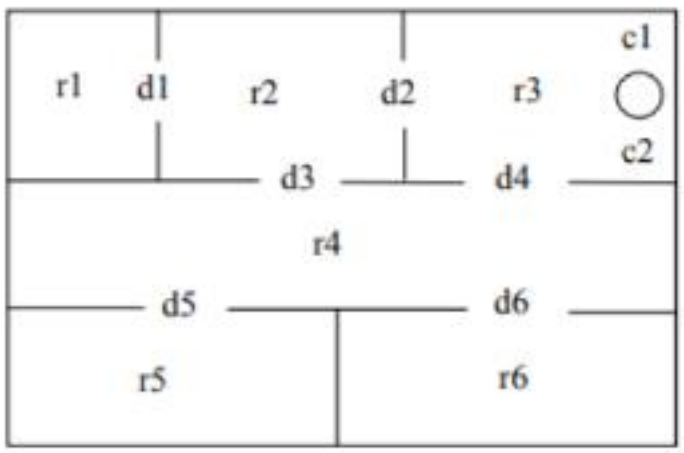

Fig - The Robot's Environment

3.5 Human-Robot Interface

- Human-Robot Interaction (HRI) is a field of study dedicated to understanding, designing, and evaluating robotic systems for use by or with humans. Interaction, by definition, requires communication between robots and humans.
- Communication between a human and a robot may take several forms, but these forms are largely influenced by whether the human and the robot are in close proximity to each other or not. Thus, communication and, therefore, interaction can be separated into two general categories:

➢ Remote interaction – The human and the robot are not co-located and are separated spatially or even temporally (for example, the Mars Rovers are