HISTORY OF ARTIFICIAL INTELIGENCE

Maturation of Artificial Intelligence (1943-1952)

- **Year 1943:** The first work which is now recognized as AI was done by Warren McCulloch and Walter pits in 1943. They proposed a model of **artificial neurons**.
- **Year 1949:** Donald Hebb demonstrated an updating rule for modifying the connection strength between neurons. His rule is now called **Hebbian learning**.
- **Year 1950:** The Alan Turing who was an English mathematician and pioneered Machine learning in 1950. Alan Turing publishes **"Computing Machinery and Intelligence"** in which he proposed a test. The test can check the machine's ability to exhibit intelligent behavior equivalent to human intelligence, called a **Turing test**.

The birth of Artificial Intelligence (1952-1956)

- **Year 1955:** An Allen Newell and Herbert A. Simon created the "first artificial intelligence program"Which was named as **"Logic Theorist"**. This program had proved 38 of 52 Mathematics theorems, and find new and more elegant proofs for some theorems.
- **Year 1956:** The word "Artificial Intelligence" first adopted by American Computer scientist John McCarthy at the Dartmouth Conference. For the first time, AI coined as an academic field.

At that time high-level computer languages such as FORTRAN, LISP, or COBOL were invented. And the enthusiasm for AI was very high at that time.

The golden years-Early enthusiasm (1956-1974)

- **Year 1966:** The researchers emphasized developing algorithms which can solve mathematical problems. Joseph Weizenbaum created the first chatbot in 1966, which was named as ELIZA.

Year 1972: The first intelligent humanoid robot was built in Japan which was named as WABOT-1.

he golden years-Early enthusiasm (1956-1974)

- **Year 1966:** The researchers emphasized developing algorithms which can solve mathematical problems. Joseph Weizenbaum created the first chatbot in 1966, which was named as ELIZA.

- **Year 1972:** The first intelligent humanoid robot was built in Japan which was named as WABOT-1.

The first AI winter (1974-1980)

- The duration between years 1974 to 1980 was the first AI winter duration. AI winter refers to the time period where computer scientist dealt with a severe shortage of funding from government for AI researches.
- During AI winters, an interest of publicity on artificial intelligence was decreased
-
- A boom of AI (1980-1987)

- **Year 1980:** After AI winter duration, AI came back with "Expert System". Expert systems were programmed that emulate the decision-making ability of a human expert.
- In the Year 1980, the first national conference of the American Association of Artificial Intelligence **was held at Stanford University**.

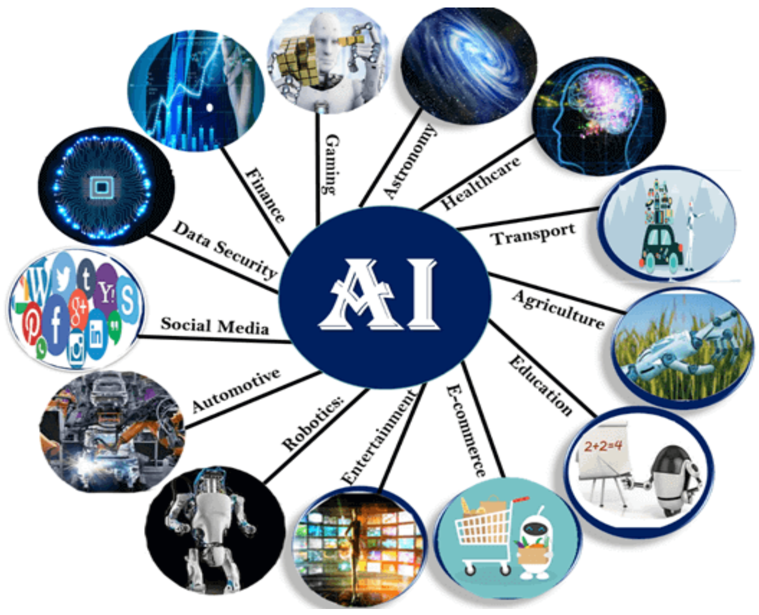
The emergence of intelligent agents (1993-2011)

- **Year 1997:** In the year 1997, IBM Deep Blue beats world chess champion, Gary Kasparov, and became the first computer to beat a world chess champion.
- **Year 2002:** for the first time, AI entered the home in the form of Roomba, a vacuum cleaner.
- **Year 2006:** AI came in the Business world till the year 2006. Companies like Facebook, Twitter, a

THE STATE OF ART IN AI

Artificial Intelligence has various applications in today's society. It is becoming essential for today's time because it can solve complex problems with an efficient way in multiple industries, such as Healthcare, entertainment, finance, education, etc. AI is making our daily life more comfortable and fast.

Following are some sectors which have the application oF Artificial Intelligence:



1. AI in Astronomy

- Artificial Intelligence can be very useful to solve complex universe problems. AI technology can be helpful for understanding the universe such as how it works, origin, etc.

2. AI in Healthcare

- In the last, five to ten years, AI becoming more advantageous for the healthcare industry and going to have a significant impact on this industry.
- Healthcare Industries are applying AI to make a better and faster diagnosis than humans. AI can help doctors with diagnoses and can inform when patients are worsening so that medical help can reach to the patient before hospitalization.

3. AI in Gaming

- AI can be used for gaming purpose. The AI machines can play strategic games like chess, where the machine needs to think of a large number of possible places.

4. AI in Finance

- AI and finance industries are the best matches for each other. The finance industry is implementing automation, chatbot, adaptive intelligence, algorithm trading, and machine learning into financial processes.

5. AI in Data Security

- The security of data is crucial for every company and cyber-attacks are growing very rapidly in the digital world. AI can be used to make your data more safe and secure. Some examples such as AEG bot, AI2 Platform,are used to determine software bug and cyber-attacks in a better way.

6. AI in Social Media

- Social Media sites such as Facebook, Twitter, and Snapchat contain billions of user profiles, which need to be stored and managed in a very efficient way. AI can organize and manage massive amounts of data. AI can analyze lots of data to identify the latest trends, hashtag, and requirement of different users.

7. AI in Travel & Transport

- AI is becoming highly demanding for travel industries. AI is capable of doing various travel related works such as from making travel arrangement to suggesting the hotels, flights, and best routes to the customers. Travel industries are using AI-powered chatbots which can make human-like interaction with customers for better and fast response.

8. AI in Automotive Industry

- Some Automotive industries are using AI to provide virtual assistant to their user for better performance. Such as Tesla has introduced TeslaBot, an intelligent virtual assistant.
- Various Industries are currently working for developing self-driven cars which can make your journey more safe and secure.

9. AI in Robotics:

- Artificial Intelligence has a remarkable role in Robotics. Usually, general robots are programmed such that they can perform some repetitive task, but with the help of AI, we can create intelligent robots which can perform tasks with their own experiences without pre-programmed.
- Humanoid Robots are best examples for AI in robotics, recently the intelligent Humanoid robot named as Erica and Sophia has been developed which can talk and behave like humans.

10. AI in Entertainment

- We are currently using some AI based applications in our daily life with some entertainment services such as Netflix or Amazon. With the help of ML/AI algorithms, these services show the recommendations for programs or shows.

11. AI in Agriculture

- Agriculture is an area which requires various resources, labor, money, and time for best result. Now a day's agriculture is becoming digital, and AI is emerging in this field. Agriculture is applying AI as agriculture robotics, solid and crop monitoring, predictive analysis. AI in agriculture can be very helpful for farmers.

12. AI in E-commerce

- AI is providing a competitive edge to the e-commerce industry, and it is becoming more demanding in the e-commerce business. AI is helping shoppers to discover associated products with recommended size, color, or even brand.

13. AI in education:

- AI can automate grading so that the tutor can have more time to teach. AI chatbot can communicate with students as a teaching assistant.
- AI in the future can be work as a personal virtual tutor for students, which will be accessible easily at any time and any place.

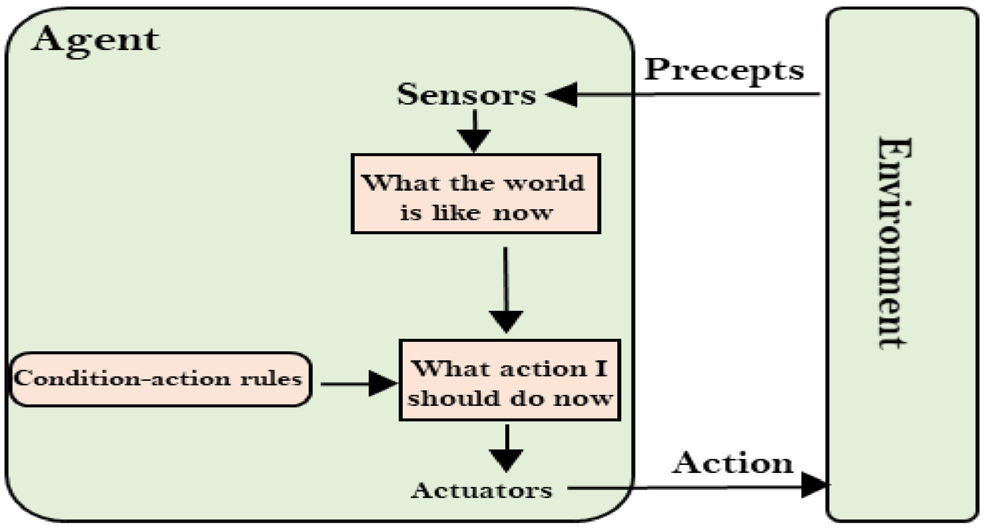
STRUCTURE OF AGENTS

Agents can be grouped into five classes based on their degree of perceived intelligence and capability. All these agents can improve their performance and generate better action over the time. These are given below:

- Simple Reflex Agent
- Model-based reflex agent
- Goal-based agents
- Utility-based agent
- Learning agent

1. Simple Reflex agent:

- The Simple reflex agents are the simplest agents. These agents take decisions on the basis of the current percepts and ignore the rest of the percept history.
- These agents only succeed in the fully observable environment.
- The Simple reflex agent does not consider any part of percepts history during their decision and action process.
- The Simple reflex agent works on Condition-action rule, which means it maps the current state to action. Such as a Room Cleaner agent, it works only if there is dirt in the room.
- Problems for the simple reflex agent design approach:
    - They have very limited intelligence
    - They do not have knowledge of non-perceptual parts of the current state
    - Mostly too big to generate and to store.
    - Not adaptive to changes in the environment.

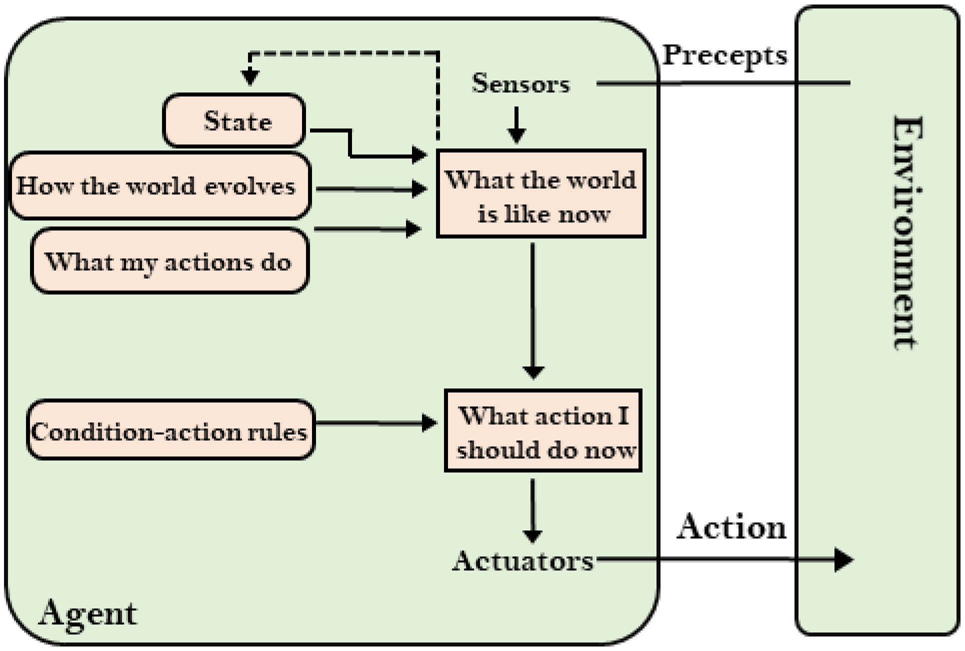


2. Model-based reflex agent

- The Model-based agent can work in a partially observable environment, and track the situation.
- A model-based agent has two important factors:
    - **Model:** It is knowledge about "how things happen in the world," so it is called a Model-based agent.
    - **Internal State:** It is a representation of the current state based on percept history.
- These agents have the model, "which is knowledge of the world" and based on the model they perform actions.
- Updating the agent state requires information about:

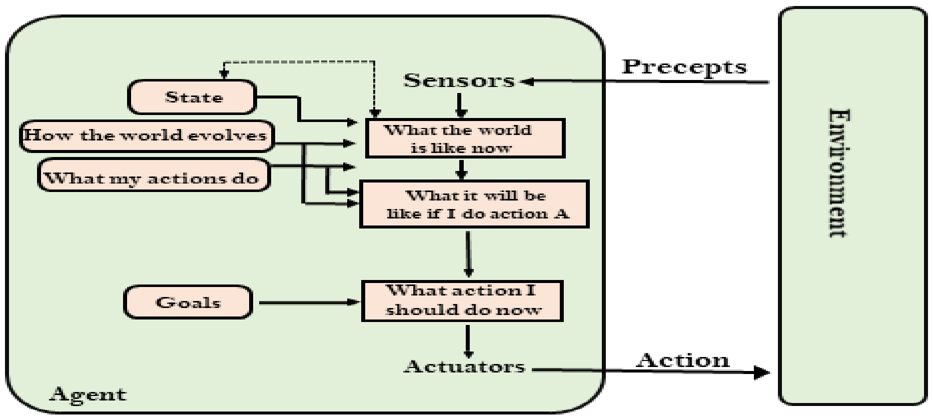
a. How the world evolves

b. How the agent's action affects the world.



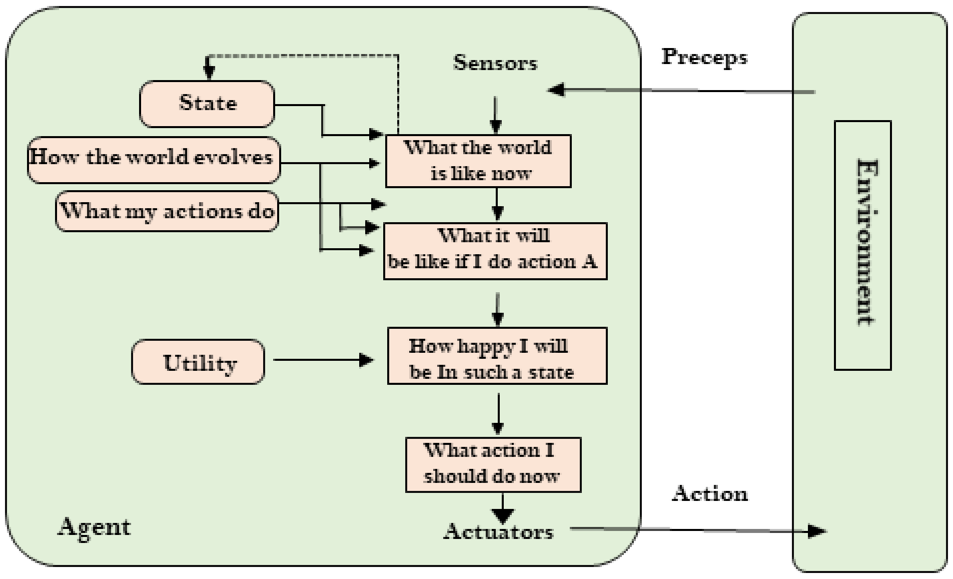
3. Goal-based agents

- The knowledge of the current state environment is not always sufficient to decide for an agent to what to do.
- The agent needs to know its goal which describes desirable situations.
- Goal-based agents expand the capabilities of the model-based agent by having the "goal" information.
- They choose an action, so that they can achieve the goal.
- These agents may have to consider a long sequence of possible actions before deciding whether the goal is achieved or not. Such considerations of different scenario are called searching and planning, which makes an agent proactive.



4. Utility-based agents

- These agents are similar to the goal-based agent but provide an extra component of utility measurement which makes them different by providing a measure of success at a given state.
- Utility-based agent act based not only goals but also the best way to achieve the goal.
- The Utility-based agent is useful when there are multiple possible alternatives, and an agent has to choose in order to perform the best action.
- The utility function maps each state to a real number to check how efficiently each action achieves the goals.


Sensors
Preceps
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Utility
How happy I will be In such a state
What action I should do now
Agent
Actuators
Action
Environment

5. Learning Agents

- A learning agent in AI is the type of agent which can learn from its past experiences, or it has learning capabilities.
- It starts to act with basic knowledge and then able to act and adapt automatically through learning.
- A learning agent has mainly four conceptual components, which are:

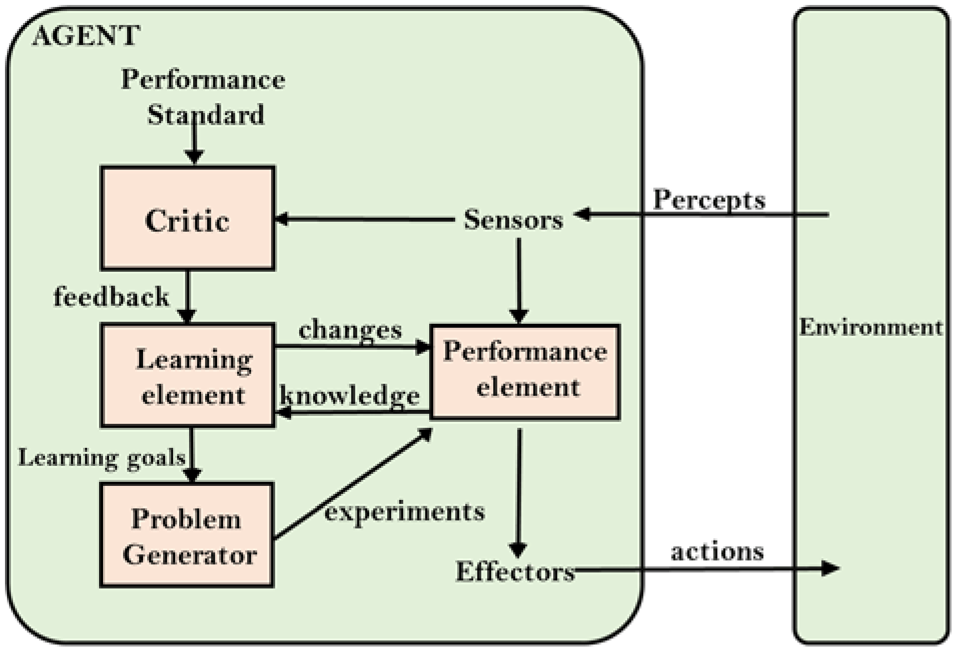
a. **Learning element:** It is responsible for making improvements by learning from environment

b. **Critic:** Learning element takes feedback from critic which describes that how well the agent is doing with respect to a fixed performance standard.

c. **Performance element:** It is responsible for selecting external action

d. **Problem generator:** This component is responsible for suggesting actions that will lead to new and informative experiences.

Hence, learning agents are able to learn, analyze performance, and look for new ways to improve the performance.



Agents in Artificial Intelligence

An AI system can be defined as the study of the rational agent and its environment. The agents sense the environment through sensors and act on their environment through actuators. An AI agent can have mental properties such as knowledge, belief, intention, etc.

What is an Agent?

An agent can be anything that perceiveits environment through sensors and act upon that environment through actuators. An Agent runs in the cycle of **perceiving**, **thinking**, and **acting**. An agent can be:

- **Human-Agent:** A human agent has eyes, ears, and other organs which work for sensors and hand, legs, vocal tract work for actuators.
- **Robotic Agent:** A robotic agent can have cameras, infrared range finder, NLP for sensors and various motors for actuators.
- **Software Agent:** Software agent can have keystrokes, file contents as sensory input and act on those inputs and display output on the screen.

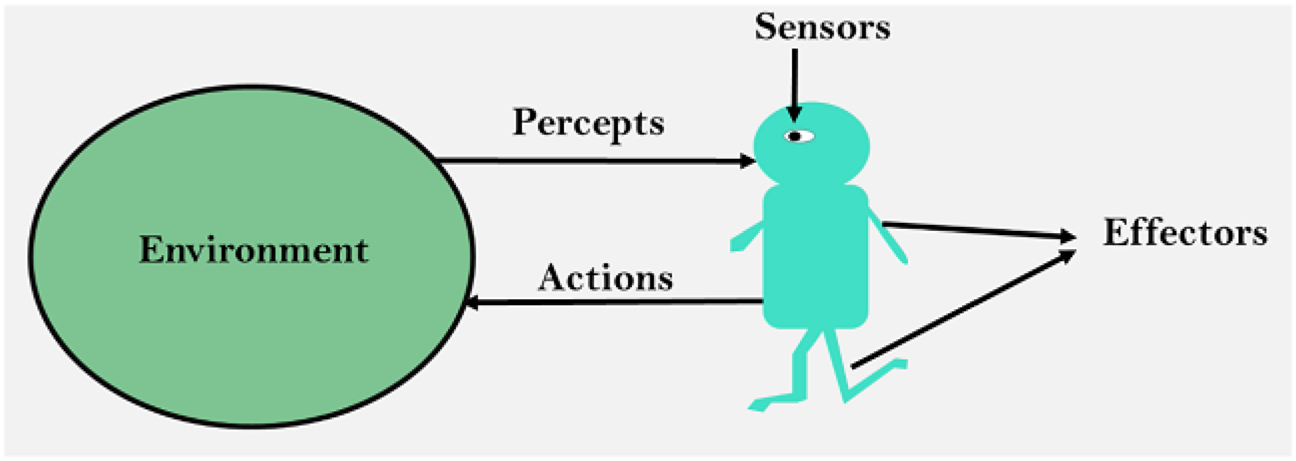
Hence the world around us is full of agents such as thermostat, cellphone, camera, and even we are also agents.

Before moving forward, we should first know about sensors, effectors, and actuators.

Sensor: Sensor is a device which detects the change in the environment and sends the information to other electronic devices. An agent observes its environment through sensors.

Actuators: Actuators are the component of machines that converts energy into motion. The actuators are only responsible for moving and controlling a system. An actuator can be an electric motor, gears, rails, etc.

Effectors: Effectors are the devices which affect the environment. Effectors can be legs, wheels, arms, fingers, wings, fins, and display screen.



Intelligent Agents:

An intelligent agent is an autonomous entity which act upon an environment using sensors and actuators for achieving goals. An intelligent agent may learn from the environment to achieve their goals. A thermostat is an example of an intelligent agent.

Following are the main four rules for an AI agent:

- **Rule 1:** An AI agent must have the ability to perceive the environment.
- **Rule 2:** The observation must be used to make decisions.
- **Rule 3:** Decision should result in an action.
- **Rule 4:** The action taken by an AI agent must be a rational action.

Rational Agent:

A rational agent is an agent which has clear preference, models uncertainty, and acts in a way to maximize its performance measure with all possible actions.

A rational agent is said to perform the right things. AI is about creating rational agents to use for game theory and decision theory for various real-world scenarios.

For an AI agent, the rational action is most important because in AI reinforcement learning algorithm, for each best possible action, agent gets the positive reward and for each wrong action, an agent gets a negative reward.

Note: Rational agents in AI are very similar to intelligent agents.

Rationality:

The rationality of an agent is measured by its performance measure. Rationality can be judged on the basis of following points:

- Performance measure which defines the success criterion.
- Agent prior knowledge of its environment.
- Best possible actions that an agent can perform.
- The sequence of percepts.

Note: Rationality differs from Omniscience because an Omniscient agent knows the actual outcome of its action and act accordingly, which is not possible in reality.

Structure of an AI Agent

The task of AI is to design an agent program which implements the agent function. The structure of an intelligent agent is a combination of architecture and agent program. It can be viewed as:

1. Agent = Architecture + Agent program

Following are the main three terms involved in the structure of an AI agent:

Architecture: Architecture is machinery that an AI agent executes on.

Agent Function: Agent function is used to map a percept to an action.

1. f:P* → A

Agent program: Agent program is an implementation of agent function. An agent program executes on the physical architecture to produce function f.

Agent Environment in AI

An environment is everything in the world which surrounds the agent, but it is not a part of an agent itself. An environment can be described as a situation in which an agent is present.

The environment is where agent lives, operate and provide the agent with something to sense and act upon it. An environment is mostly said to be non-feministic.

NAATURE of Environment

As per Russell and Norvig, an environment can have various features from the point of view of an agent:

1. Fully observable vs Partially Observable
2. Static vs Dynamic
3. Discrete vs Continuous
4. Deterministic vs Stochastic
5. Single-agent vs Multi-agent
6. Episodic vs sequential
7. Known vs Unknown
8. Accessible vs Inaccessible

1. Fully observable vs Partially Observable:

- If an agent sensor can sense or access the complete state of an environment at each point of time then it is **a fully observable** environment, else it is **partially observable**.
- A fully observable environment is easy as there is no need to maintain the internal state to keep track history of the world.
- An agent with no sensors in all environments then such an environment is called as **unobservable**.

2. Deterministic vs Stochastic:

- If an agent's current state and selected action can completely determine the next state of the environment, then such environment is called a deterministic environment.
- A stochastic environment is random in nature and cannot be determined completely by an agent.
- In a deterministic, fully observable environment, agent does not need to worry about uncertainty.

3. Episodic vs Sequential:

- In an episodic environment, there is a series of one-shot actions, and only the current percept is required for the action.
- However, in Sequential environment, an agent requires memory of past actions to determine the next best actions.

4. Single-agent vs Multi-agent

- If only one agent is involved in an environment, and operating by itself then such an environment is called single agent environment.
- However, if multiple agents are operating in an environment, then such an environment is called a multi-agent environment.
- The agent design problems in the multi-agent environment are different from single agent environment.

5. Static vs Dynamic:

- If the environment can change itself while an agent is deliberating then such environment is called a dynamic environment else it is called a static environment.
- Static environments are easy to deal because an agent does not need to continue looking at the world while deciding for an action.
- However for dynamic environment, agents need to keep looking at the world at each action.
- Taxi driving is an example of a dynamic environment whereas Crossword puzzles are an example of a static environment.

6. Discrete vs Continuous:

- If in an environment there are a finite number of percepts and actions that can be performed within it, then such an environment is called a discrete environment else it is called continuous environment.
- A chess gamecomes under discrete environment as there is a finite number of moves that can be performed.
- A self-driving car is an example of a continuous environment.

7. Known vs Unknown

- Known and unknown are not actually a feature of an environment, but it is an agent's state of knowledge to perform an action.
- In a known environment, the results for all actions are known to the agent. While in unknown environment, agent needs to learn how it works in order to perform an action.
- It is quite possible that a known environment to be partially observable and an Unknown environment to be fully observable.

8. Accessible vs Inaccessible

- If an agent can obtain complete and accurate information about the state's environment, then such an environment is called an Accessible environment else it is called inaccessible.
- An empty room whose state can be defined by its temperature is an example of an accessible environment.
- Information about an event on earth is an example of Inaccessible environment.
-

Unit 2

Search algorithms are one of the most important areas of Artificial Intelligence. This topic will explain all about the search algorithms in AI.

Problem-solving agents:

In Artificial Intelligence, Search techniques are universal problem-solving methods. **Rational agents** or **Problem-solving agents** in AI mostly used these search strategies or algorithms to solve a specific problem and provide the best result. Problem-solving agents are the goal-based agents and use atomic representation. In this topic, we will learn various problem-solving search algorithms.

Search Algorithm Terminologies:

- **Search:** Searchingis a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:
  a. **Search Space:** Search space represents a set of possible solutions, which a system may have.
  b. **Start State:** It is a state from where agent begins **the search**.
  c. **Goal test:** It is a function which observe the current state and returns whether the goal state is achieved or not.

Search tree: A tree representation of search problem is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.
Actions: It gives the description of all the available actions to the agent.
Transition model: A description of what each action do, can be represented as a transition model.
Path Cost: It is a function which assigns a numeric cost to each path.
Solution: It is an action sequence which leads from the start node to the goal node.
Optimal Solution: If a solution has the lowest cost among all solutions.

Properties of Search Algorithms:

Following are the four essential properties of search algorithms to compare the efficiency of these algorithms:

Completeness: A search algorithm is said to be complete if it guarantees to return a solution if at least any solution exists for any random input.

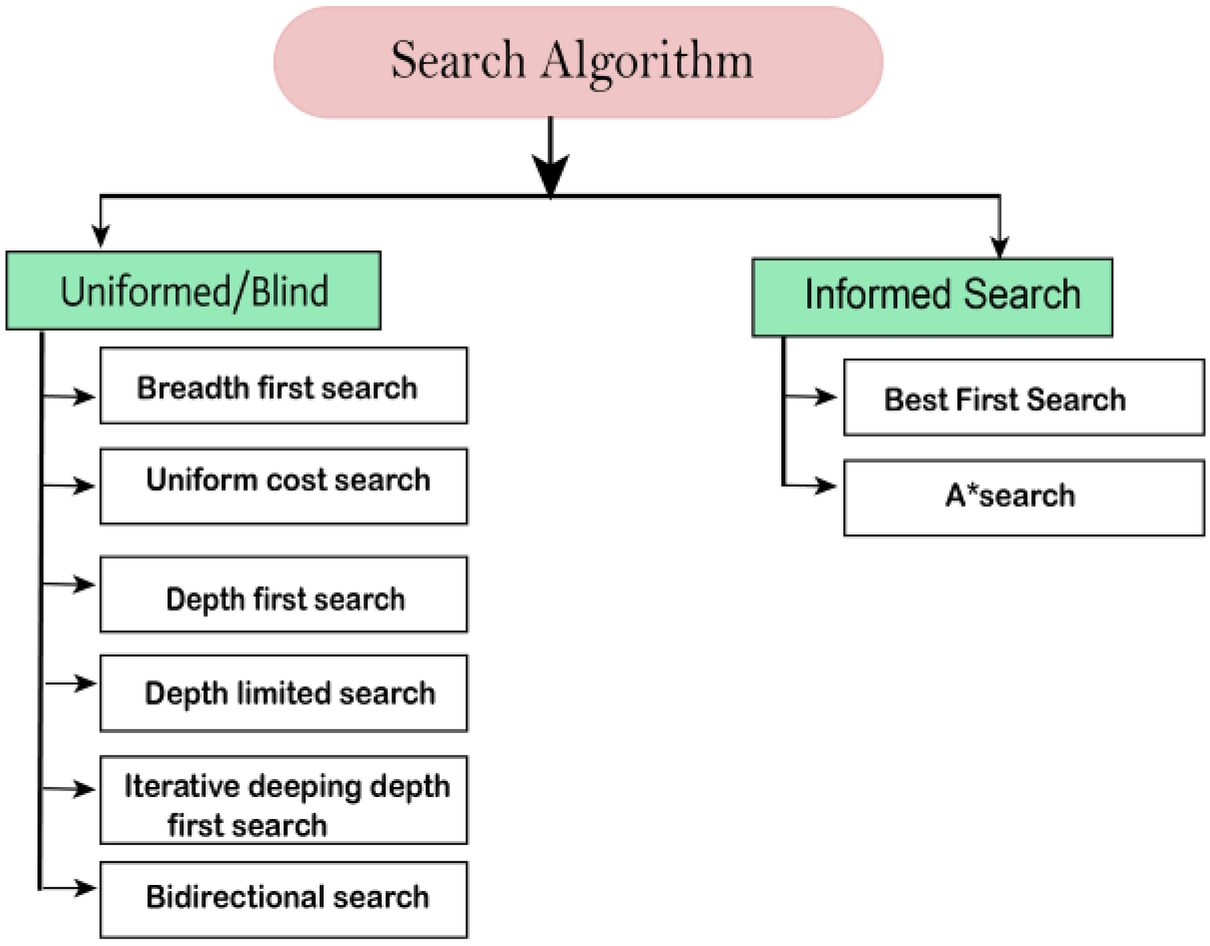
Optimality: If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution for is said to be an optimal solution.

Time Complexity: Time complexity is a measure of time for an algorithm to complete its task.

Space Complexity: It is the maximum storage space required at any point during the search, as the complexity of the problem.

Types of search algorithms

Based on the search problems we can classify the search algorithms into uninformed (Blind search) search and informed search (Heuristic search) algorithms.



Uninformed/Blind Search:

The uninformed search does not contain any domain knowledge such as closeness, the location of the goal.

It operates in a brute-force way as it only includes information about how to traverse the tree and how to identify leaf and goal nodes.

Uninformed search applies a way in which search tree is searched without any information about the search space like initial state operators and test for the goal, so it is also called blind search.

It examines each node of the tree until it achieves the goal node.

It can be divided into five main types:

Breadth-first search

Uniform cost search

Depth-first search

Iterative deepening depth-first search

Bidirectional Search

Informed Search

Informed search algorithms use domain knowledge. In an informed search, problem information is available which can guide the search.

Informed search strategies can find a solution more efficiently than an uninformed search strategy. Informed search is also called a Heuristic search.

A heuristic is a way which might not always be guaranteed for best solutions but guaranteed to find a good solution in reasonable time.

Informed search can solve much complex problem which could not be solved in another way.

An example of informed search algorithms is a traveling salesman problem.

1. Greedy Search
2. A* Search

Uninformed Search Algorithms

Uninformed search is a class of general-purpose search algorithms which operates in brute force-way. Uninformed search algorithms do not have additional information about state or search space other than how to traverse the tree, so it is also called blind search.

Following are the various types of uninformed search algorithms:

1. **Breadth-first Search**
2. **Depth-first Search**
3. **Depth-limited Search**
4. **Iterative deepening depth-first search**
5. **Uniform cost search**
6. **Bidirectional Search**

1. Breadth-first Search:

- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.
- The breadth-first search algorithm is an example of a general-graph search algorithm.
- Breadth-first search implemented using FIFO queue data structure.

Advantages:

- BFS will provide a solution if any solution exists.
- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

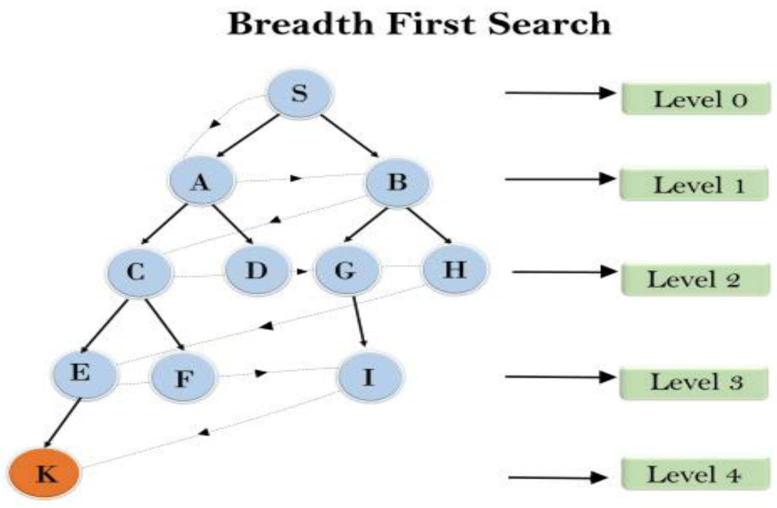
Disadvantages:

- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.

Example:

In the below tree structure, we have shown the traversing of the tree using BFS algorithm from the root node S to goal node K. BFS search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

1. S---> A--->B---->C--->D---->G--->H--->E---->F---->I---->K



Time Complexity: Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d= depth of shallowest solution and b is a node at every state.

T (b) = $1+b^2+b^3+.......+ b^d= O (b^d)$

Space Complexity: Space complexity of BFS algorithm is given by the Memory size of frontier which is O(b^d).

Completeness: BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.

Optimality: BFS is optimal if path cost is a non-decreasing function of the depth of the node.

2. Depth-first Search:

- Depth-first search isa recursive algorithm for traversing a tree or graph data structure.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation.
- The process of the DFS algorithm is similar to the BFS algorithm.

Note: Backtracking is an algorithm technique for finding all possible solutions using recursion.

Advantage:

- DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.
- It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

Disadvantage:

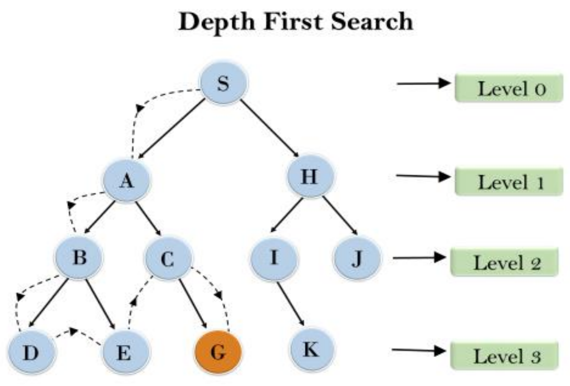
- There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
- DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

Example:

In the below search tree, we have shown the flow of depth-first search, and it will follow the order as:

Root node--->Left node ----> right node.

It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.



Completeness: DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.

Time Complexity: Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

$T(n)= 1+ n^2+ n^3 +.........+ n^m=O(n^m)$

Where, m= maximum depth of any node and this can be much larger than d (Shallowest solution depth)

Space Complexity: DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is **O(bm)**.

Optimal: DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.

3. Depth-Limited Search Algorithm:

A depth-limited search algorithm is similar to depth-first search with a predetermined limit. Depth-limited search can solve the drawback of the infinite path in the Depth-first search. In this algorithm, the node at the depth limit will treat as it has no successor nodes further.

Depth-limited search can be terminated with two Conditions of failure:

- Standard failure value: It indicates that problem does not have any solution.
- Cutoff failure value: It defines no solution for the problem within a given depth limit.

Advantages:

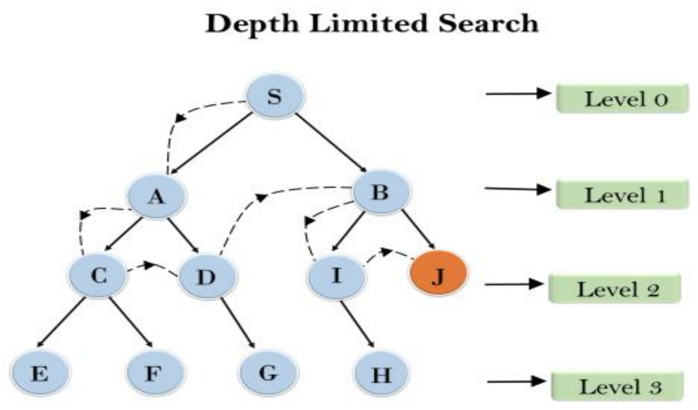
Depth-limited search is Memory efficient.

Disadvantages:

- Depth-limited search also has a disadvantage of incompleteness.

- It may not be optimal if the problem has more than one solution.

Example:



Completeness: DLS search algorithm is complete if the solution is above the depth-limit.

Time Complexity: Time complexity of DLS algorithm is **$O(b^{\ell})$**.

Space Complexity: Space complexity of DLS algorithm is $O(b \times \ell)$.

Optimal: Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if $\ell > d$.

4. Uniform-cost Search Algorithm:

Uniform-cost search is a searching algorithm used for traversing a weighted tree or graph. This algorithm comes into play when a different cost is available for each edge. The primary goal of the uniform-cost search is to find a path to the goal node which has the lowest cumulative cost. Uniform-cost search expands nodes according to their path costs form the root node. It can be used to solve any graph/tree where the optimal cost is in demand. A uniform-cost search algorithm is implemented by the priority queue. It gives maximum priority to the lowest cumulative cost. Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.

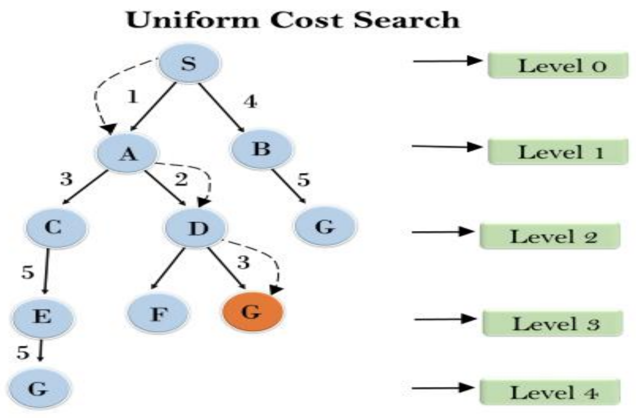
Advantages:

- Uniform cost search is optimal because at every state the path with the least cost is chosen.

Disadvantages:

- It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.

Example:



Completeness:

Uniform-cost search is complete, such as if there is a solution, UCS will find it.

Time Complexity:

Let C* **is Cost of the optimal solution**, and **ε** is each step to get closer to the goal node. Then the number of steps is = C*/ε+1. Here we have taken +1, as we start from state 0 and end to C*/ε.

Hence, the worst-case time complexity of Uniform-cost search is**$O(b^{1 + [C^*/\varepsilon]})$/**.

Space Complexity:

The same logic is for space complexity so, the worst-case space complexity of Uniform-cost search is **$O(b^{1 + [C^*/\varepsilon]})$**.

Optimal:

Uniform-cost search is always optimal as it only selects a path with the lowest path cost.

5. Iterative deepeningdepth-first Search:

The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.

This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.

This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.

The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

Advantages:

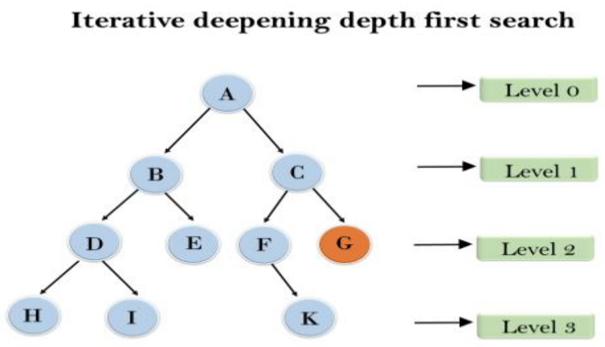
- Itcombines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

Disadvantages:

- The main drawback of IDDFS is that it repeats all the work of the previous phase.

Example:

Following tree structure is showing the iterative deepening depth-first search. IDDFS algorithm performs various iterations until it does not find the goal node. The iteration performed by the algorithm is given



as:

1'stIteration----->A
2'ndIteration---->A,B,C
3'rdIteration------>A,B,D,E,C,F,G
4'thIteration------>A,B,D,H,I,E,C,F,K,G
In the fourth iteration, the algorithm will find the goal node.

Completeness:

This algorithm is complete is ifthe branching factor is finite.

Time Complexity:

Let's suppose b is the branching factor and depth is d then the worst-case time complexity is **O(b^d)**.

Space Complexity:

The space complexity of IDDFS will be **O(bd)**.

Optimal:

IDDFS algorithm is optimal if path cost is a non- decreasing function of the depth of the node.

6. Bidirectional Search Algorithm:

Bidirectional search algorithm runs two simultaneous searches, one form initial state called as forward-search and other from goal node called as backward-search, to find the goal node.

Bidirectional search replaces one single search graph with two small subgraphs in which one starts the search from an initial vertex and other starts from goal vertex.

The search stops when these two graphs intersect each other.

Bidirectional search can use search techniques such as BFS, DFS, DLS, etc.

Advantages:

- Bidirectional search is fast.
- Bidirectional search requires less memory

Disadvantages:

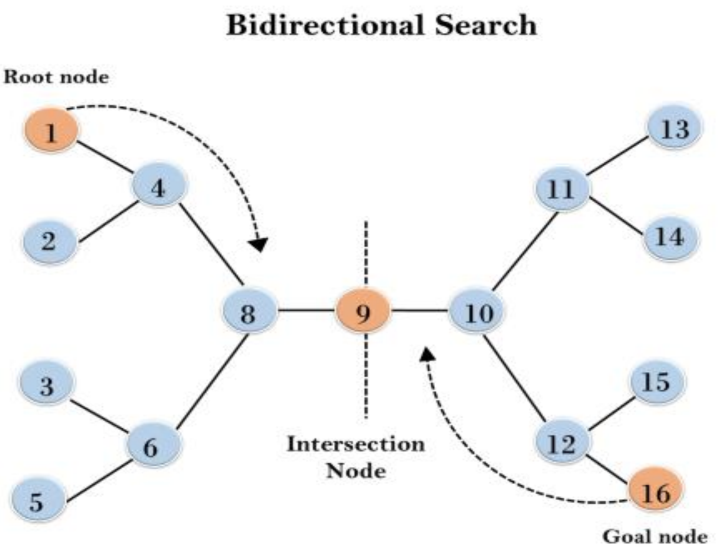
- Implementation of the bidirectional search tree is difficult.
- **In bidirectional search, one should know the goal state in advance.**

Example:

In the below search tree, bidirectional search algorithm is applied.

This algorithm divides one graph/tree into two sub-graphs. It starts traversing from node 1 in the forward direction and starts from goal node 16 in the backward direction.

The algorithm terminates at node 9 where two searches meet.



Completeness: Bidirectional Search is complete if we use BFS in both searches.

Time Complexity: Time complexity of bidirectional search using BFS is **$O(b^d)$**.

Space Complexity: Space complexity of bidirectional search is **$O(b^d)$**.

Optimal: Bidirectional search is Optimal.

Informed Search Algorithms

Informed search algorithm contains an array of knowledge such as how far we are from the goal, path cost, how to reach to goal node, etc. This knowledge help agents to explore less to the search space and find more efficiently the goal node.

The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.

Heuristics function: Heuristic is a function which is used in Informed Search, and it finds the most promising path.

It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal.

The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time.

Heuristic function estimates how close a state is to the goal. It is represented by h(n), and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.

Admissibility of the heuristic function is given as:

1. h(n) <= h*(n)

Here h(n) is heuristic cost, and h*(n) is the estimated cost. Hence heuristic cost should be less than or equal to the estimated cost.

Pure Heuristic Search:

Pure heuristic search is the simplest form of heuristic search algorithms. It expands nodes based on their heuristic value h(n).

It maintains two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.

On each iteration, each node n with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list. The algorithm continues unit a goal state is found.

In the informed search we will discuss two main algorithms which are given below:

Best First Search Algorithm(Greedy search)

A* Search Algorithm

1.) Best-first Search Algorithm (Greedy Search):

Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms.

It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function,

1. f(n)= g(n).

Were, h(n)= estimated cost from node n to the goal.

The greedy best first algorithm is implemented by the priority queue.

Best first search algorithm:

- **Step 1:** Place the starting node into the OPEN list.
- **Step 2:** If the OPEN list is empty, Stop and return failure.
- **Step 3:** Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.
- **Step 4:** Expand the node n, and generate the successors of node n.
- **Step 5:** Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
- **Step 6:** For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
- **Step 7:** Return to Step 2.

Advantages:

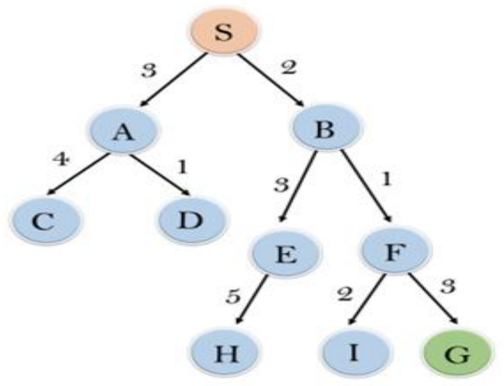
- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

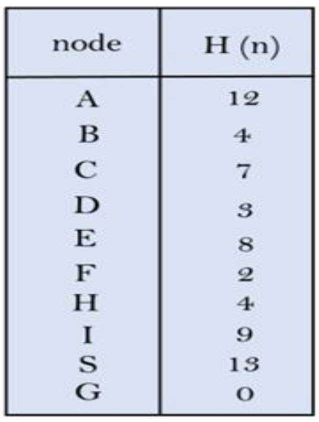
Disadvantages:

- It can behave as an unguided depth-first search in the worst case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal.

Example:

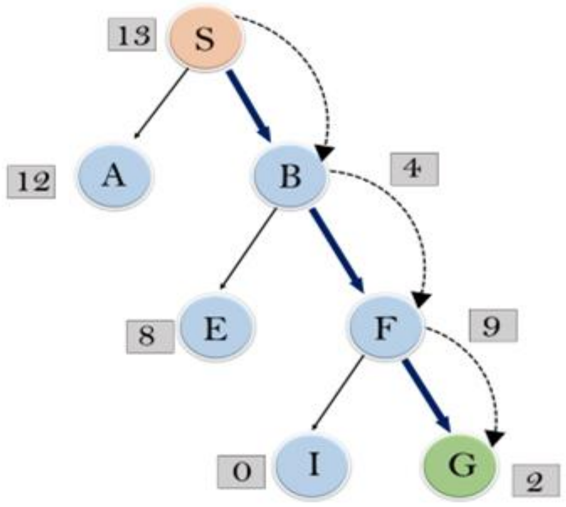
Consider the below search problem, and we will traverse it using greedy best-first search. At each iteration, each node is expanded using evaluation function f(n)=h(n) , which is given in the below table.





node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

In this search example, we are using two lists which are **OPEN** and **CLOSED** Lists. Following are the iteration for traversing the above example.



Expand the nodes of S and put in the CLOSED list

Initialization: Open [A, B], Closed [S]

Iteration 1: Open [A], Closed [S, B]

Iteration2: Open[E,F,A],Closed[S,B]
: Open [E, A], Closed [S, B, F]

Iteration3: Open[I,G,E,A],Closed[S,B,F]
: Open [I, E, A], Closed [S, B, F, G]

Hence the final solution path will be: **S----> B----->F----> G**

Time Complexity: The worst case time complexity of Greedy best first search is O(b^m).

Space Complexity: The worst case space complexity of Greedy best first search is O(b^m). Where, m is the maximum depth of the search space.

Complete: Greedy best-first search is also incomplete, even if the given state space is finite.

Optimal: Greedy best first search algorithm is not optimal.

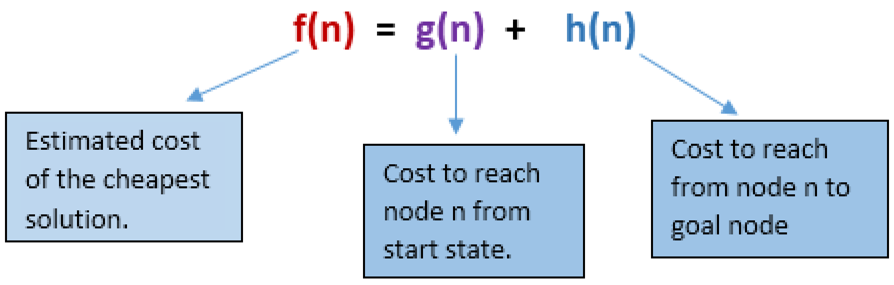
2.) A* Search Algorithm:

A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n).

It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function.

This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).

In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.


f(n) = g(n) + h(n)
Estimated cost of the cheapest solution.
Cost to reach node n from start state.
Cost to reach from node n to goal node

At each point in the search space, only those node is expanded which have the lowest value of f(n), and the algorithm terminates when the goal node is found.

Algorithm of A* search:

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to **Step 2**.

Advantages:

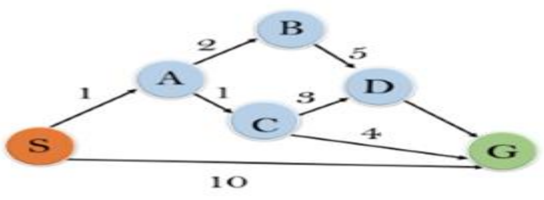
- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

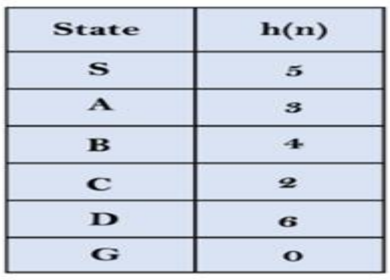
Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

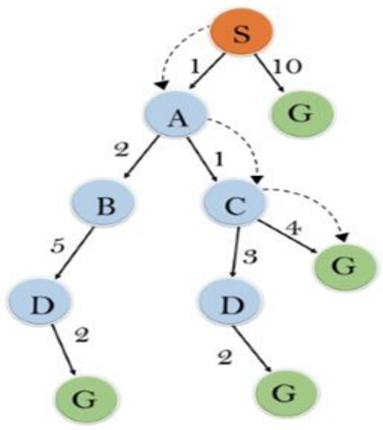
Example:

In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state.
Here we will use OPEN and CLOSED list.





State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0



Solution:

Initialization: {(S, 5)}

Iteration1: {(S--> A, 4), (S-->G, 10)}

Iteration2: {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}

Iteration3: {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}

Iteration 4 will give the final result, as **S--->A--->C--->G** it provides the optimal path with cost 6.

Points to remember:

- A* algorithm returns the path which occurred first, and it does not search for all remaining paths.
- The efficiency of A* algorithm depends on the quality of heuristic.
- A* algorithm expands all nodes which satisfy the condition f(n)<="" li="">

Complete: A* algorithm is complete as long as:

- Branching factor is finite.
- Cost at every action is fixed.

Optimal: A* search algorithm is optimal if it follows below two conditions:

- **Admissible:** the first condition requires for optimality is that h(n) should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.
- **Consistency:** Second required condition is consistency for only A* graph-search.

If the heuristic function is admissible, then A* tree search will always find the least cost path.

Time Complexity: The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So the time complexity is O(b^d), where b is the branching factor.

Space Complexity: The space complexity of A* search algorithm is **O(b^d)**

unit-3

Part-2 **logical agent**

** 5M 2M

*) **Knowledge based agent in AI :-**

→ An intelligent agent needs knowledge about the real world for taking decisions & 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 & take actions.

→ knowledge-based agents are composed of two main parts.

*) knowledge-base

*) Inference system

knowledge based agent must able to following:

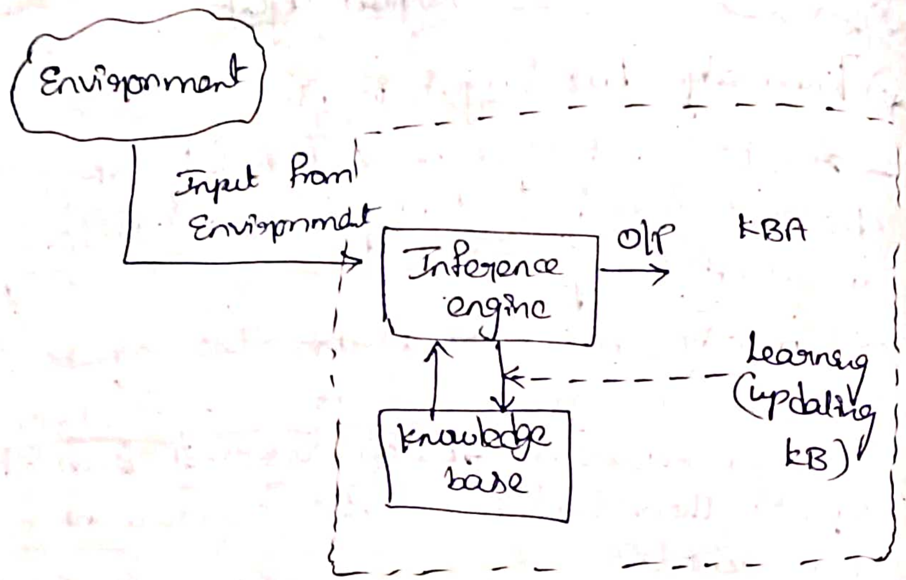
→ An agent should be able to represent states, actions etc.

→ An agent should be able to incorporate new percepts

→ An agent can update the internal representation of the world

→ An agent can appropriate actions.

The architecture of knowledge-based agent



*) Is representing a generalized architecture for a knowledge-based agent

*) The knowledge based agent (KBA) take input from the environment by perceiving the environment.

*) The input is taken by inference engine of the agent & which communicate with KB as per as knowledge store in KB.

*) The learning element of KBA Regularly updates the KB by new knowledge.

Knowledge base :-

*) knowledge base is a central component of a knowledge-based agent it is called KB.

*) These sentences are expressed in a language which is called a knowledge representation language.

*) The knowledge-base of KBA stores the world.

Inference system :-

*) Inference means deriving new sentences from old.

*) Inference system allows us to add a new sentence to the knowledge base.

*) Inference system applies logical rules to the KB to new information

*) The system generate with new facts & the agent can update the KB.

*) An inference system works into two rules they are

*) Forward chaining

*) Backward chaining

*) The wumpus world in AI :-

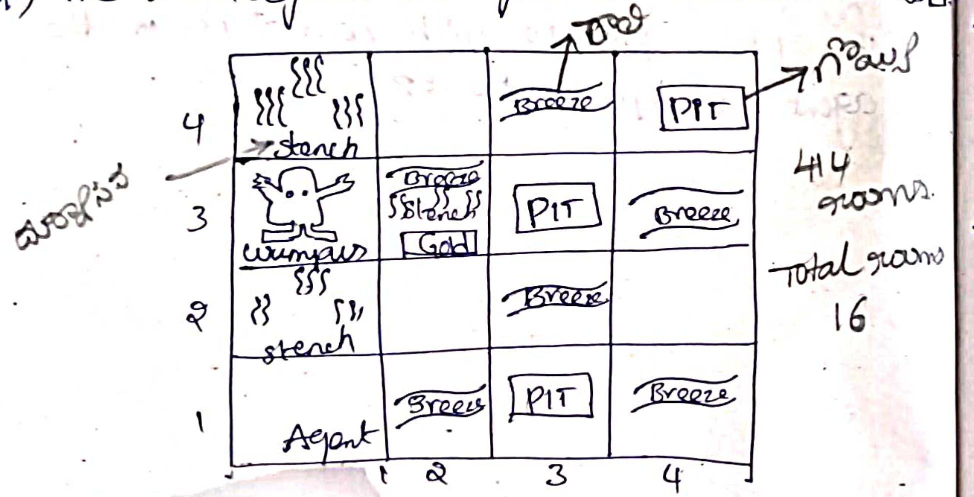
10M A simple world to illustrate the worth of a knowledge-based agent & to represent knowledge representation.

*) It was inspired by a video game "Hunt the wumpus by Gregory Yob in 1973."

*) The wumpus world has 4*4 rooms connected with passageways.

*) The total 16 rooms which are connected with each other.

*) The knowledge-based agent is forward in world.



*) The rooms adjacent to the wumpus rooms are smelly it have some stench

*) The room adjacent to PIT has a breeze if the agent reaches to near PIT.

*) There will be glitter in the room and if the room has gold.

*) The wumpus can be killed by the agent if the agent is facing.

Performance measure :-

*) +1000 reward points if the agent comes out of the cave with the gold

*) -1000 points penalty for being eaten by wumpus or falling into the pit.

*) -1 for each action & -10 for using an arrow.

*) The game ends if either agent dies.

Environment :-

*) 4*4 grid of rooms

*) The agent ~~in~~ room square [1,1], facing initially toward the right.

*) Location of wumpus and gold are chosen randomly except with first square [1,1].

*) Each square of the cave can be pit with prob. of 0.2 except in first square.

Actuators :-

→ Left turn
→ Right turn
→ Move forward
→ Grab - తీస్కో
→ Release
→ shoot

Sensors :-

*) The agent will perceive the stench if he is in the room adjacent to the wumpus.

*) The agent will perceive the breeze if the room directly adjacent to the pit.

*) The agent will perceive the glitter in the room of gold is present.

*) The agent will perceive the bump if the walks into a wall.

*) when the wumpus is shot it emits a horrible scream which can be perceived anywhere in the cave.

*) In ~~which~~ different indicators in each sensor

*) Eg :- If agent perceives stench, breeze, but no glitter, no bump and no scream then it can be represented as
[stench, Breeze, None, None, None]

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 ok	2,2	3,3	3,4
1,1 A OK	2,1 OK	3,1	4,1

Room is safe No stench No Breeze

A-agent
B-agent
G-Glitter Gold
ok-safe, square
P-pit
S-stench
V-visited
W = wumpus

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 ok	2,2 P?	3,2	4,2
1,1 ok	2,1 A OK	3,1 P?	4,1

Perceived breeze adjacent room is not safe Go back

*) Propositional logic : A Very simple logic :-
Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions.
→ The statement which is either true or false.
→ It is a technique knowledge representation in logical & mathematical form.

Eg :- *) It is sunday (True)
*) The sun rises from west (false)
*) 5 is prime number (True)
*) 3+3=7 (false)

Some basic facts about propositional logic:-

*) propositional logic is also called boolean alogic and it is works on 0 & 1.

*) we can use symbol for a representing a proposition such as A, B, C, P, Q, R etc.

*) propositional either true or false but it cannot be both.

*) propositional logic consists of an object, relations or function & logical connectives.

*) The connectives are also called logical operators

*) logical operator connectives with two sentences.

*) A proposition formula which is always true is called tautology.

*) A proposition formula which is always false is called tautology.

*) the statements which are questions, commands, or opinions are not proposition such as "where is rohini", "How are you", "what is your name" are not propositions.

Syntax :-
Two types of propositions they are.
*) Atomic propositions
*) Compound "

Atomic :-
→ It is simple propositions
→ The sentence must be either true or false

Eg:- 2+2 =4 (True)
The sun is cold (false)

Compound :-
The constructed by combining with simpler or atomic propositions by using logical connectives.

Eg:- 1*) "It is raining today & street is wet"

*) propositional theorem proving :-

Inference :-
we need intelligent computers which can create new logic from old logic or by evidence so generating the conclusion from in evidence & facts to termed as Inference.

Inference rule :-
*) Templates for generating in valid arguments
*) Inference rules are applied to derive proofs in AI.
*) A sequence of the conclusion that leads to the desired goal.

Implication :- It is one of the logical connectives which can be represented as $P \to Q$

*) It is boolean expression.

Converse :-

*) The right-hand side proposition goes to the left-hand side and vice-versa.

*) It can be written as $Q \rightarrow P$

Contraposition :-

*) The negation of converse is termed as contrapositive. It is represented as $\neg Q \rightarrow \neg P$.

Inverse :-

*) The negation of implication is called Inverse

*) It is represented as $\neg P \rightarrow \neg Q$.

P	Q	$P \rightarrow Q$	$Q \rightarrow P$	$\neg P$	$\neg Q$	$\neg Q \rightarrow \neg P$	$\neg P \rightarrow \neg Q$
T	T	T	T	F	F	T	T
T	F	F	T	F	T	F	T
F	T	T	F	T	F	T	F
F	F	T	T	T	T	T	T

Types of Inference rules :-

1. **Modus Ponens :-**

$$\boxed{\frac{P \rightarrow Q,\ P}{\therefore Q}}$$

Eg :- I am sleepy then I go to bed $\Rightarrow P \rightarrow Q$

I am sleepy $\Rightarrow P$

I go to bed $\Rightarrow Q$

2. **Modus Tollens :-**

$$\boxed{\frac{P \rightarrow Q,\ \sim Q}{\sim P}}$$

Eg :-

"I am sleepy then I go to bed" $\Rightarrow P \rightarrow Q$

"I do not go to the bed" $\Rightarrow \sim Q$

"I am not sleepy" $\Rightarrow \sim P$

3. **Hypothetical syllogism :-**

If $P \rightarrow R$ is true whenever $P \rightarrow Q$ is true & $Q \rightarrow R$ is true.

Eg :-

→ If you have my home key then you can unlock my home $P \rightarrow Q$

→ If you unlock my home then you can take my money $Q \rightarrow R$.

→ If you have my home key then you can take my money $P \rightarrow R$.

4. **Disjunctive syllogism :-**

$$\boxed{\frac{P \vee Q,\ \neg P}{Q}}$$

Eg :- Today is sunday or monday $\Rightarrow P \vee Q$

Today is not sunday $\Rightarrow \neg P$

Today is Monday $\Rightarrow Q$

5. **Addition :-**

$$\boxed{\frac{P}{P \vee Q}}$$

Eg :- I have a vanilla ice-cream $\Rightarrow P$

I have chocolate "

I have vanilla or chocolate ice-cream $(P \vee Q)$

6. Simplification :-

$$\boxed{\frac{P \wedge Q}{Q} \text{ (or) } \frac{P \wedge Q}{P}}$$

7. Resolution :-

$$\boxed{\frac{P \vee Q, \neg P \wedge R}{Q \vee R}}$$

Effective propositional Model checking agents based on propositional logic

unit-3

Constraint satisfication problem :-

*) so many techniques like local search, Adversarial search to solve different problems.

*) The objective of every problem solving technique is one.

*) To find a solution to reach the goal.

*) No constraints on the agents while solving the problems & reaching to its solutions.

*) we will discuss another type of problem solving technique known as Constraint satisfication problem.

*) It is understood that constraint satisfies means solving a problem under certain constraints or rules.

*) Technique is understanding to the problem structure as well as complexity.

Constraint satisfaction depends on three Components :-

x : It is a set of variable

D : It is a set of domains
There is a specific domain for each variable

c : set of constraints.

*) It Constraint satisfaction domain the space of problem specific constraints.

*) The Constraint value consists of a pair of (scope, rel)

*) The scope of variable which participate in the Constraint.

Solving Constraint satisfication problem :-

*) A state-space

*) The notion of the solution

*) A state space is assigning value to some all Variables.

($x_1 = v_1, x_2 = v_2$ and so on....)

assignment of values to a variable in three ways:

Consistent or legal Assignment :-

An assignment which does not violate any Constraint or rule is called Consistent or legal Assignment.

Complete Assignment :-

An assignment where every Variable is assigned with a Value & the solution to the csp remains Constraints.

Partial assignment :-

An assignment which assign values to some of the variables only.

Types of domain in csp :-

Discrete domain :- It is a Infinite domain which can have one state for multiple Variable.

finite domain :- It is a finite domain which can Continuous states describing one domain for one specific variable.

Types of Constraint in csp :-

unary Constraints	:-	(single variable)
Binary "	:-	(Two Variable)
Global "	:-	Arbitrary

Algorithm are used to solve the following type of Constraint :-

Linear Constraint : each variable Containing an integer value exists in linear form only.

Non-linear Constraint : Are used in non-linear programming where each variable exists in a non-linear form.

Constraint propagation

→ we can search for a solution

→ we can perform a special type of inference is Called Constraint propagation.

→ Idea behind Constraint propagation is local Consistency.

→ Variables are treated as nodes & each binary Constraint is treated as an arc in the given problem.

Node consistency :- A single variable is said to be node consistent if all the values in the variable domain.

Arc consistency :- If every value in its domain satisfies the binary constraints of the variable.

Path consistency :- A set of two variable with respect to a third variable can be extended over another variable

K-consistency :- It is used to define the notation stronger forms of propagation.

Inference in csps -

Backtracking search for csps.

pcc

Unit _4

First-Order Logic in Artificial intelligence

In the topic of Propositional logic, we have seen that how to represent statements using propositional logic. But unfortunately, in propositional logic, we can only represent the facts, which are either true or false. PL is not sufficient to represent the complex sentences or natural language statements. The propositional logic has very limited expressive power. Consider the following sentence, which we cannot represent using PL logic.

- **"Some humans are intelligent", or**
- **"Sachin likes cricket."**

To represent the above statements, PL logic is not sufficient, so we required some more powerful logic, such as first-order logic.

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, wumpus, ...
  - **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, ...
- As a natural language, first-order logic also has two main parts:

Syntax

a. **Semantics**

Syntax of First-Order logic:

The syntax of FOL determines which collection of symbols is a logical expression in first-order logic. The basic syntactic elements of first-order logic are symbols. We write statements in short-hand notation in FOL.

Basic Elements of First-order logic:

Following are the basic elements of FOL syntax:

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf, ...
Connectives	∧, ∨, ¬, ⇒, ⇔
Equality	==
Quantifier	∀, ∃

Atomic sentences:

- Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.
- We can represent atomic sentences as **Predicate (term1, term2, ..., term n)**.

Example: Ravi and Ajay are brothers: => Brothers(Ravi, Ajay).
Chinky is a cat: => cat (Chinky).

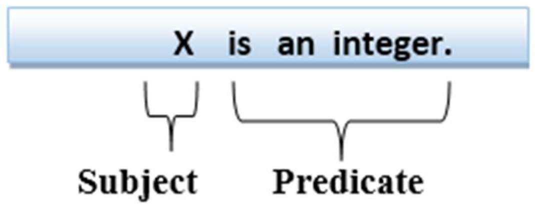
Complex Sentences:

- Complex sentences are made by combining atomic sentences using connectives.

First-order logic statements can be divided into two parts:

- **Subject:** Subject is the main part of the statement.
- **Predicate:** A predicate can be defined as a relation, which binds two atoms together in a statement.

Consider the statement: "x is an integer.", it consists of two parts, the first part x is the subject of the statement and second part "is an integer," is known as a predicate.



Quantifiers in First-order logic:

- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
- These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifier:

a. **Universal Quantifier, (for all, everyone, everything)**

b. **Existential quantifier, (for some, at least one).**

Universal Quantifier:

Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing.

The Universal quantifier is represented by a symbol ∀, which resembles an inverted A.

Note: In universal quantifier we use implication "→".

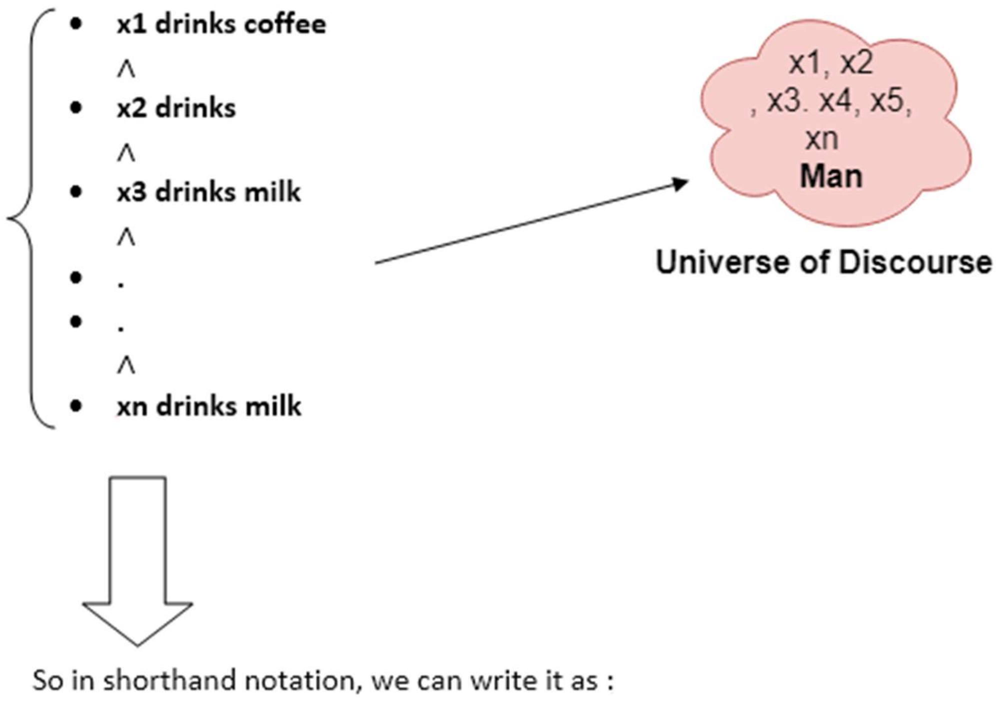
If x is a variable, then ∀x is read as:

- **For all x**
- **For each x**
- **For every x.**

Example:

All man drink coffee.

Let a variable x which refers to a cat so all x can be represented in UOD as below:



∀x man(x) → drink (x, coffee).

It will be read as: There are all x where x is a man who drink coffee.

Existential Quantifier:

Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something.

It is denoted by the logical operator ∃, which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

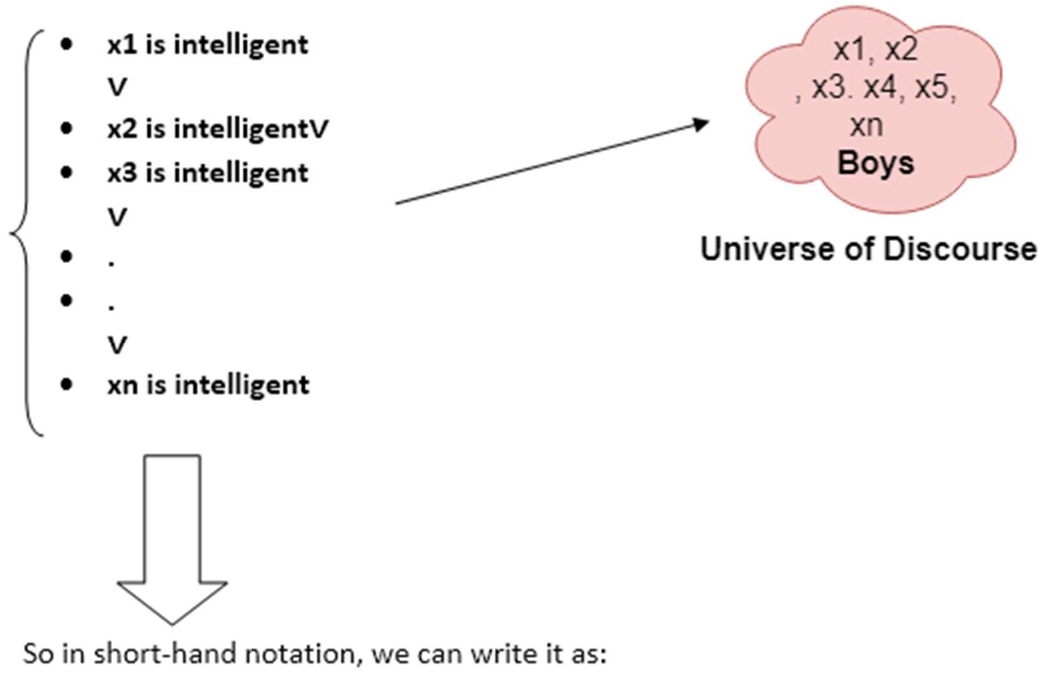
Note: In Existential quantifier we always use AND or Conjunction symbol (∧).

If x is a variable, then existential quantifier will be ∃x or ∃(x). And it will be read as:

- **There exists a 'x.'**
- **For some 'x.'**
- **For at least one 'x.'**

Example:

Some boys are intelligent.



∃x: boys(x) ∧ intelligent(x)

It will be read as: There are some x where x is a boy who is intelligent.

Points to remember:

- The main connective for universal quantifier **∀** is implication **→**.
- The main connective for existential quantifier **∃** is and **∧**.

Properties of Quantifiers:

- In universal quantifier, ∀x∀y is similar to ∀y∀x.
- In Existential quantifier, ∃x∃y is similar to ∃y∃x.
- ∃x∀y is not similar to ∀y∃x.

Some Examples of FOL using quantifier:

1. All birds fly.
In this question the predicate is "**fly(bird)**."
And since there are all birds who fly so it will be represented as follows.
∀x bird(x) →fly(x).

2. Every man respects his parent.
In this question, the predicate is "**respect(x, y)," where x=man, and y= parent**.
Since there is every man so will use ∀, and it will be represented as follows:
∀x man(x) → respects (x, parent).

3. Some boys play cricket.
In this question, the predicate is "**play(x, y)**," where x= boys, and y= game. Since there are some boys so we will use ∃**, and it will be represented as**:

∃x boys(x) → play(x, cricket).

4. Not all students like both Mathematics and Science.
In this question, the predicate is "**like(x, y)," where x= student, and y= subject**.
Since there are not all students, so we will use **∀ with negation, so** following representation for this:

¬∀ (x) [student(x) → like(x, Mathematics) ∧ like(x, Science)].

5. Only one student failed in Mathematics.
In this question, the predicate is "**failed(x, y)," where x= student, and y= subject**.
Since there is only one student who failed in Mathematics, so we will use following representation for this:

∃(x) [student(x) → failed (x, Mathematics) ∧∀ (y) [¬(x==y) ∧ student(y) → ¬failed (x, Mathematics)].

Free and Bound Variables:

The quantifiers interact with variables which appear in a suitable way. There are two types of variables in First-order logic which are given below:

Free Variable: A variable is said to be a free variable in a formula if it occurs outside the scope of the quantifier.

Example: ∀x ∃(y)[P (x, y, z)], where z is a free variable.

Bound Variable: A variable is said to be a bound variable in a formula if it occurs within the scope of the quantifier

Example: ∀x [A (x) B(y)], here x and y are the bound variables.

Knowledge Engineering in First-order logic

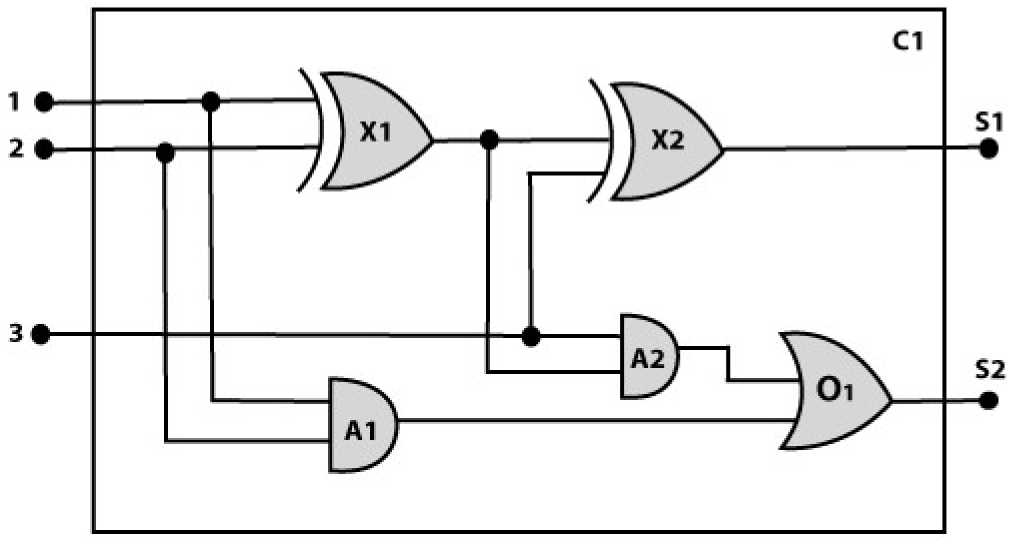
What is knowledge-engineering?

The process of constructing a knowledge-base in first-order logic is called as knowledge-engineering. In **knowledge-engineering**, someone who investigates a particular domain, learns important concept of that domain, and generates a formal representation of the objects, is known as **knowledge engineer**.

In this topic, we will understand the Knowledge engineering process in an electronic circuit domain, which is already familiar. This approach is mainly suitable for creating **special-purpose knowledge base**.

The knowledge-engineering process:

Following are some main steps of the knowledge-engineering process. Using these steps, we will develop a knowledge base which will allow us to reason about digital circuit (**One-bit full adder**) which is given below



1. Identify the task:

The first step of the process is to identify the task, and for the digital circuit, there are various reasoning tasks.

At the first level or highest level, we will examine the functionality of the circuit:

- **Does the circuit add properly?**
- **What will be the output of gate A2, if all the inputs are high?**

At the second level, we will examine the circuit structure details such as:

- **Which gate is connected to the first input terminal?**
- **Does the circuit have feedback loops?**

2. Assemble the relevant knowledge:

In the second step, we will assemble the relevant knowledge which is required for digital circuits. So for digital circuits, we have the following required knowledge:

- Logic circuits are made up of wires and gates.
- Signal flows through wires to the input terminal of the gate, and each gate produces the corresponding output which flows further.

- In this logic circuit, there are four types of gates used: **AND, OR, XOR, and NOT**.
- All these gates have one output terminal and two input terminals (except NOT gate, it has one input terminal).

3. Decide on vocabulary:

The next step of the process is to select functions, predicate, and constants to represent the circuits, terminals, signals, and gates.

Firstly we will distinguish the gates from each other and from other objects. Each gate is represented as an object which is named by a constant, such as, **Gate(X1)**. The functionality of each gate is determined by its type, which is taken as constants such as **AND, OR, XOR, or NOT**. Circuits will be identified by a predicate: **Circuit (C1)**.

For the terminal, we will use predicate: **Terminal(x)**.

For gate input, we will use the function **In(1, X1)** for denoting the first input terminal of the gate, and for output terminal we will use **Out (1, X1)**.

The function **Arity(c, i, j)** is used to denote that circuit c has i input, j output.

The connectivity between gates can be represented by predicate **Connect(Out(1, X1), In(1, X1))**.

We use a unary predicate **On (t)**, which is true if the signal at a terminal is on.

4. Encode general knowledge about the domain:

To encode the general knowledge about the logic circuit, we need some following rules:

- If two terminals are connected then they have the same input signal, it can be represented as:

1. ∀ t1, t2 Terminal (t1) ∧ Terminal (t2) ∧ Connect (t1, t2) → Signal (t1) = Signal (2).
    - Signal at every terminal will have either value 0 or 1, it will be represented as:

1. ∀ t Terminal (t) →Signal (t) = 1 ∨Signal (t) = 0.
    - Connect predicates are commutative:

1. ∀ t1, t2 Connect(t1, t2) → Connect (t2, t1).
    - Representation of types of gates:

1. ∀ g Gate(g) ∧ r = Type(g) → r = OR ∨r = AND ∨r = XOR ∨r = NOT.
    - Output of AND gate will be zero if and only if any of its input is zero

∀ g Gate(g) ∧ Type(g) = AND →Signal (Out(1, g))= 0 ⇔ ∃n Signal (In(n, g))= 0.

- Output of OR gate is 1 if and only if any of its input

1. ∀ g Gate(g) ∧ Type(g) = OR → Signal (Out(1, g))= 1 ⇔ ∃n Signal (In(n, g))= 1

- Output of XOR gate is 1 if and only if its inputs are different:

1. ∀ g Gate(g) ∧ Type(g) = XOR → Signal (Out(1, g)) = 1 ⇔ Signal (In(1, g)) ≠ Signal (In(2, g)).

- Output of NOT gate is invert of its input:

1. ∀ g Gate(g) ∧ Type(g) = NOT → Signal (In(1, g)) ≠ Signal (Out(1, g)).

- All the gates in the above circuit have two inputs and one output (except NOT gate).

1. ∀ g Gate(g) ∧ Type(g) = NOT → Arity(g, 1, 1)
2. ∀ g Gate(g) ∧ r =Type(g) ∧ (r= AND ∨r= OR ∨r= XOR) → Arity (g, 2, 1).

- All gates are logic circuits:

1. ∀ g Gate(g) → Circuit (g).

5. Encode a description of the problem instance:

Now we encode problem of circuit C1, firstly we categorize the circuit and its gate components. This step is easy if ontology about the problem is already thought.

This step involves the writing simple atomics sentences of instances of concepts, which is known as ontology.

For the given circuit C1, we can encode the problem instance in atomic sentences as below:

Since in the circuit there are two XOR, two AND, and one OR gate so atomic sentences for these gates will be:

1. For XOR gate: Type(x1)= XOR, Type(X2) = XOR
2. For AND gate: Type(A1) = AND, Type(A2)= AND
3. For OR gate: Type (O1) = OR.

And then represent the connections between all the gates.

Note: Ontology defines a particular theory of the nature of existence.

6. Pose queries to the inference procedure and get answers:

In this step, we will find all the possible set of values of all the terminal for the adder circuit. The first query will be:

What should be the combination of input which would generate the first output of circuit C1, as 0 and a second output to be 1?

1. ∃ i1, i2, i3 Signal (In(1, C1))=i1 ∧ Signal (In(2, C1))=i2 ∧ Signal (In(3, C1))= i3
2. ∧ Signal (Out(1, C1)) =0 ∧ Signal (Out(2, C1))=1

7. Debug the knowledge base:

Now we will debug the knowledge base, and this is the last step of the complete process. In this step, we will try to debug the issues of knowledge base.

In the knowledge base, we may have omitted assertions like 1 ≠ 0.

Inference in First-Order Logic

Inference in First-Order Logic is used to deduce new facts or sentences from existing sentences. Before understanding the FOL inference rule, let's understand some basic terminologies used in FOL.

Substitution:

Substitution is a fundamental operation performed on terms and formulas. It occurs in all inference systems in first-order logic. The substitution is complex in the presence of quantifiers in FOL. If we write **F[a/x]**, so it refers to substitute a constant "**a**" in place of variable "**x**".

Note: First-order logic is capable of expressing facts about some or all objects in the universe.

Equality:

First-Order logic does not only use predicate and terms for making atomic sentences but also uses another way, which is equality in FOL. For this, we can use **equality symbols** which specify that the two terms refer to the same object.

Example: Brother (John) = Smith.

As in the above example, the object referred by the **Brother (John)** is similar to the object referred by **Smith**. The equality symbol can also be used with negation to represent that two terms are not the same objects.

Example: ¬(x=y) which is equivalent to x ≠y.

FOL inference rules for quantifier:

As propositional logic we also have inference rules in first-order logic, so following are some basic inference rules in FOL:

- **Universal Generalization**
- **Universal Instantiation**
- **Existential Instantiation**
- **Existential introduction**

1. Universal Generalization:

- Universal generalization is a valid inference rule which states that if premise P(c) is true for any arbitrary element c in the universe of discourse, then we can have a conclusion as ∀ x P(x).
- It can be represented as: $\frac{P(c)}{\forall x\, P(x)}$.
- This rule can be used if we want to show that every element has a similar property.
- In this rule, x must not appear as a free variable.

Example: Let's represent, P(c): "**A byte contains 8 bits**", so for **∀ x P(x)** "**All bytes contain 8 bits**.", it will also be true.

2. Universal Instantiation:

- Universal instantiation is also called as universal elimination or UI is a valid inference rule. It can be applied multiple times to add new sentences.
- The new KB is logically equivalent to the previous KB.
- As per UI, **we can infer any sentence obtained by substituting a ground term for the variable**.
- The UI rule state that we can infer any sentence P(c) by substituting a ground term c (a constant within domain x) from **∀ x P(x) for any object in the universe of discourse**.
- It can be represented as: $\frac{\forall x\, P(x)}{P(c)}$.

Example:1.

IF "Every person like ice-cream"=> ∀x P(x) so we can infer that
"John likes ice-cream" => P(c)

Example: 2.

Let's take a famous example,

"All kings who are greedy are Evil." So let our knowledge base contains this detail as in the form of FOL:

∀x king(x) ∧ greedy (x) → Evil (x),

So from this information, we can infer any of the following statements using Universal Instantiation:

- **King(John) ∧ Greedy (John) → Evil (John),**
- **King(Richard) ∧ Greedy (Richard) → Evil (Richard),**
- **King(Father(John)) ∧ Greedy (Father(John)) → Evil (Father(John)),**

3. Existential Instantiation:

- Existential instantiation is also called as Existential Elimination, which is a valid inference rule in first-order logic.
- It can be applied only once to replace the existential sentence.
- The new KB is not logically equivalent to old KB, but it will be satisfiable if old KB was satisfiable.
- This rule states that one can infer P(c) from the formula given in the form of ∃x P(x) for a new constant symbol c.
- The restriction with this rule is that c used in the rule must be a new term for which P(c) is true.
- It can be represented as: $\dfrac{\exists x\, P(x)}{P(c)}$

Example:

From the given sentence: **⍰x Crown(x) ∧ OnHead(x, John),**

So we can infer: **Crown(K) ∧ OnHead(K, John),** as long as K does not appear in the knowledge base.

- The above used K is a constant symbol, which is called **Skolem constant**.
- The Existential instantiation is a special case of **Skolemization process**.

4. Existential introduction

- An existential introduction is also known as an existential generalization, which is a valid inference rule in first-order logic.
- This rule states that if there is some element c in the universe of discourse which has a property P, then we can infer that there exists something in the universe which has the property P.

- It can be represented as: $\dfrac{P(c)}{\exists x P(x)}$
- **Example: Let's say that,** "Priyanka got good marks in English." "Therefore, someone got good marks in English."

Generalized Modus Ponens Rule:

For the inference process in FOL, we have a single inference rule which is called Generalized Modus Ponens. It is lifted version of Modus ponens.

Generalized Modus Ponens can be summarized as, " P implies Q and P is asserted to be true, therefore Q must be True."

According to Modus Ponens, for atomic sentences **pi, pi', q**. Where there is a substitution θ such that SUBST **(θ, pi',) = SUBST(θ, pi)**, it can be represented as:

$$\frac{p1', p2', \ldots, pn', (p1 \land p2 \land \ldots \land pn \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

Example:

We will use this rule for Kings are evil, so we will find some x such that x is king, and x is greedy so we can infer that x is evil.

What is Unification?

- Unification is a process of making two different logical atomic expressions identical by finding a substitution. Unification depends on the substitution process.
- It takes two literals as input and makes them identical using substitution.
- Let Ψ_1 and Ψ_2 be two atomic sentences and σ be a unifier such that, $\boldsymbol{\Psi_1 = \Psi_2}$, then it can be expressed as **UNIFY(Ψ_1, Ψ_2)**.
- **Example: Find the MGU for Unify{King(x), King(John)}**

Let Ψ_1 = King(x), Ψ_2 = King(John),

Substitution θ = {John/x} is a unifier for these atoms and applying this substitution, and both expressions will be identical.

- The UNIFY algorithm is used for unification, which takes two atomic sentences and returns a unifier for those sentences (If any exist).
- Unification is a key component of all first-order inference algorithms.
- It returns fail if the expressions do not match with each other.

- The substitution variables are called Most General Unifier or MGU.

E.g. Let's say there are two different expressions, **P(x, y), and P(a, f(z))**.

In this example, we need to make both above statements identical to each other. For this, we will perform the substitution.

P(x, y)......... (i)
P(a, f(z))......... (ii)

- Substitute x with a, and y with f(z) in the first expression, and it will be represented as **a/x** and f(z)/y.
- With both the substitutions, the first expression will be identical to the second expression and the substitution set will be: **[a/x, f(z)/y]**.

Conditions for Unification:

Following are some basic conditions for unification:

- Predicate symbol must be same, atoms or expression with different predicate symbol can never be unified.
- Number of Arguments in both expressions must be identical.
- Unification will fail if there are two similar variables present in the same expression.

Forward Chaining and backward chaining in AI

In artificial intelligence, forward and backward chaining is one of the important topics, but before understanding forward and backward chaining lets first understand that from where these two terms came.

Inference engine:

The inference engine is the component of the intelligent system in artificial intelligence, which applies logical rules to the knowledge base to infer new information from known facts. The first inference engine was part of the expert system. Inference engine commonly proceeds in two modes, which are:

a. **Forward chaining**
b. **Backward chaining**

Horn Clause and Definite clause:

Horn clause and definite clause are the forms of sentences, which enables knowledge base to use a more restricted and efficient inference algorithm. Logical inference algorithms use forward and backward chaining approaches, which require KB in the form of the **first-order definite clause**.

[redacted]

Definite clause: A clause which is a disjunction of literals with **exactly one positive literal** is known as a definite clause or strict horn clause.

Horn clause: A clause which is a disjunction of literals with **at most one positive literal** is known as horn clause. Hence all the definite clauses are horn clauses.

Example: (¬ p V ¬ q V k). It has only one positive literal k.

It is equivalent to p ∧ q → k.

A. Forward Chaining

Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine. Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.

The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.

Properties of Forward-Chaining:

- It is a down-up approach, as it moves from bottom to top.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
- Forward-chaining approach is also called as data-driven as we reach to the goal using available data.
- Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

Consider the following famous example which we will use in both approaches:

Example:

"As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen."

Prove that **"Robert is criminal."**

To solve the above problem, first, we will convert all the above facts into first-order definite clauses, and then we will use a forward-chaining algorithm to reach the goal.

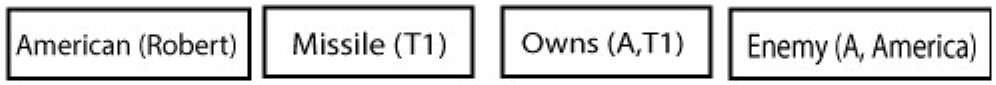
Facts Conversion into FOL:

- It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)
  American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)
- Country A has some missiles. **?p Owns(A, p) ∧ Missile(p)**. It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.
  Owns(A,T1......(2)
  Missile(T1) ....(3)
- All of the missiles were sold to country A by Robert.
  ?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) ......(4)
- Missiles are weapons.
  Missile(p) → Weapons (p) ....(5)
- Enemy of America is known as hostile.
  Enemy(p, America) →Hostile(p) ....(6)
- Country A is an enemy of America.
  Enemy (A, America) ....(7)
- Robert is American
  American(Robert). ....(8)

Forward chaining proof:

Step-1:

In the first step we will start with the known facts and will choose the sentences which do not have implications, such as: **American(Robert), Enemy(A, America), Owns(A, T1), and Missile(T1)**. All these facts will be represented as below.



Step-2:

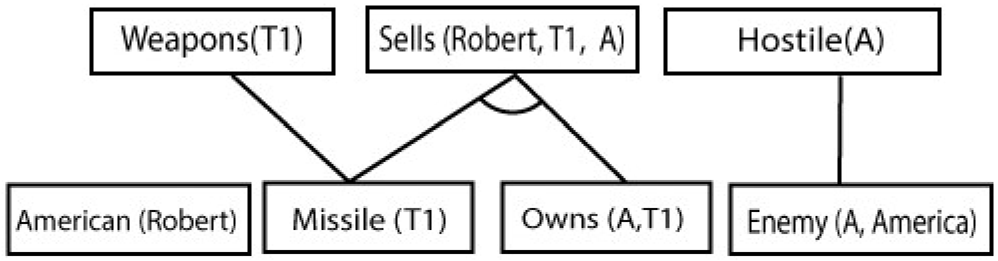
At the second step, we will see those facts which infer from available facts and with satisfied premises.

Rule-(1) does not satisfy premises, so it will not be added in the first iteration.

Rule-(2) and (3) are already added.

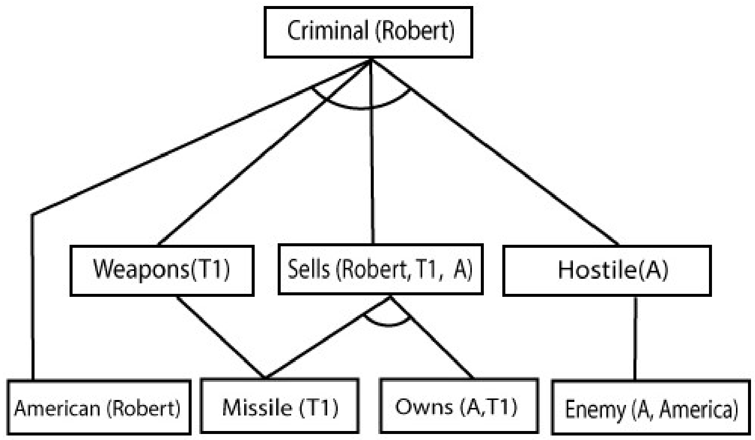
Rule-(4) satisfy with the substitution {p/T1}, **so Sells (Robert, T1, A)** is added, which infers from the conjunction of Rule (2) and (3).

Rule-(6) is satisfied with the substitution(p/A), so Hostile(A) is added and which infers from Rule-(7).



Step-3:

At step-3, as we can check Rule-(1) is satisfied with the substitution **{p/Robert, q/T1, r/A}, so we can add Criminal(Robert)** which infers all the available facts. And hence we reached our goal statement.



Hence it is proved that Robert is Criminal using forward chaining approach.

B. Backward Chaining:

Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine. A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.

Properties of backward chaining:

- It is known as a top-down approach.
- Backward-chaining is based on modus ponens inference rule.
- In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.
- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.
- Backward -chaining algorithm is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.
- The backward-chaining method mostly used a **depth-first search** strategy for proof.

Example:

In backward-chaining, we will use the same above example, and will rewrite all the rules.

- **American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)**
  Owns(A, T1) ....(2)
- **Missile(T1)**
- **?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) ......(4)**
- **Missile(p) → Weapons (p) ....(5)**
- **Enemy(p, America) →Hostile(p) ....(6)**
- **Enemy (A, America) ....(7)**
- **American(Robert). ....(8)**

Backward-Chaining proof:

In Backward chaining, we will start with our goal predicate, which is **Criminal(Robert)**, and then infer further rules.

Step-1:

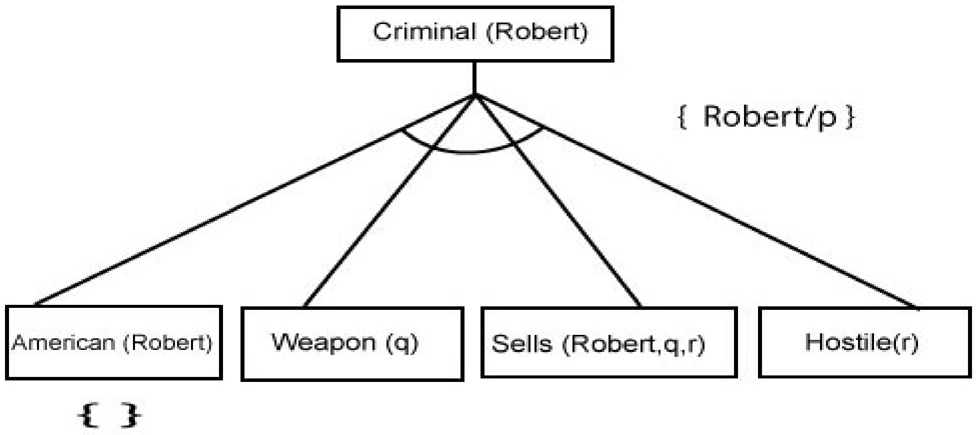
At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is "Robert is Criminal," so following is the predicate of it.

Criminal (Robert)

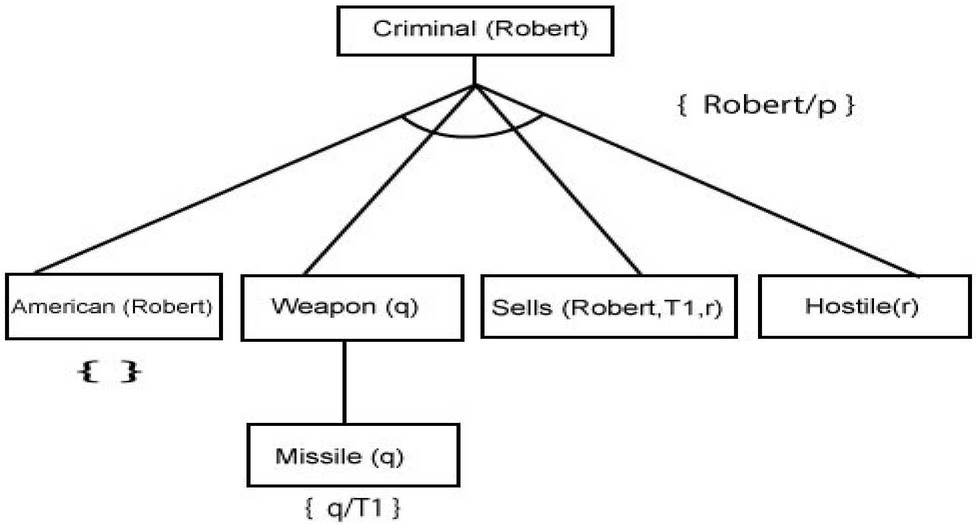
Step-2:

At the second step, we will infer other facts form goal fact which satisfies the rules. So as we can see in Rule-1, the goal predicate Criminal (Robert) is present with substitution {Robert/P}. So we will add all the conjunctive facts below the first level and will replace p with Robert.

Here we can see American (Robert) is a fact, so it is proved here.

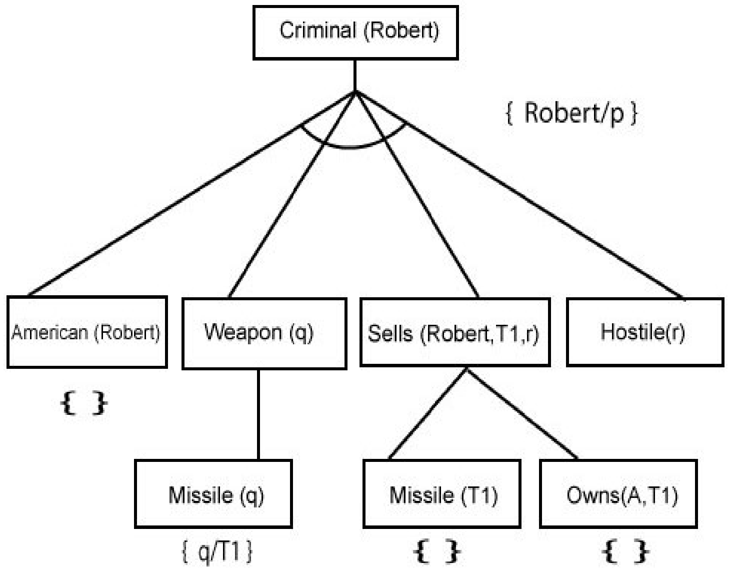


Step-3:t At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q.



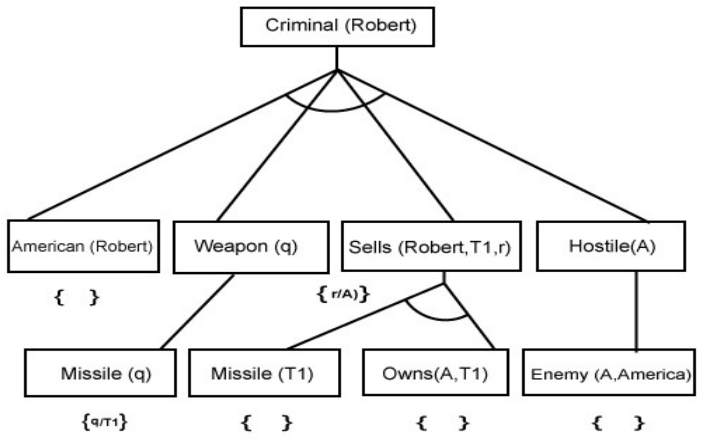
Step-4:

At step-4, we can infer facts Missile(T1) and Owns(A, T1) form Sells(Robert, T1, r) which satisfies the **Rule- 4**, with the substitution of A in place of r. So these two statements are proved here.



Step-5:

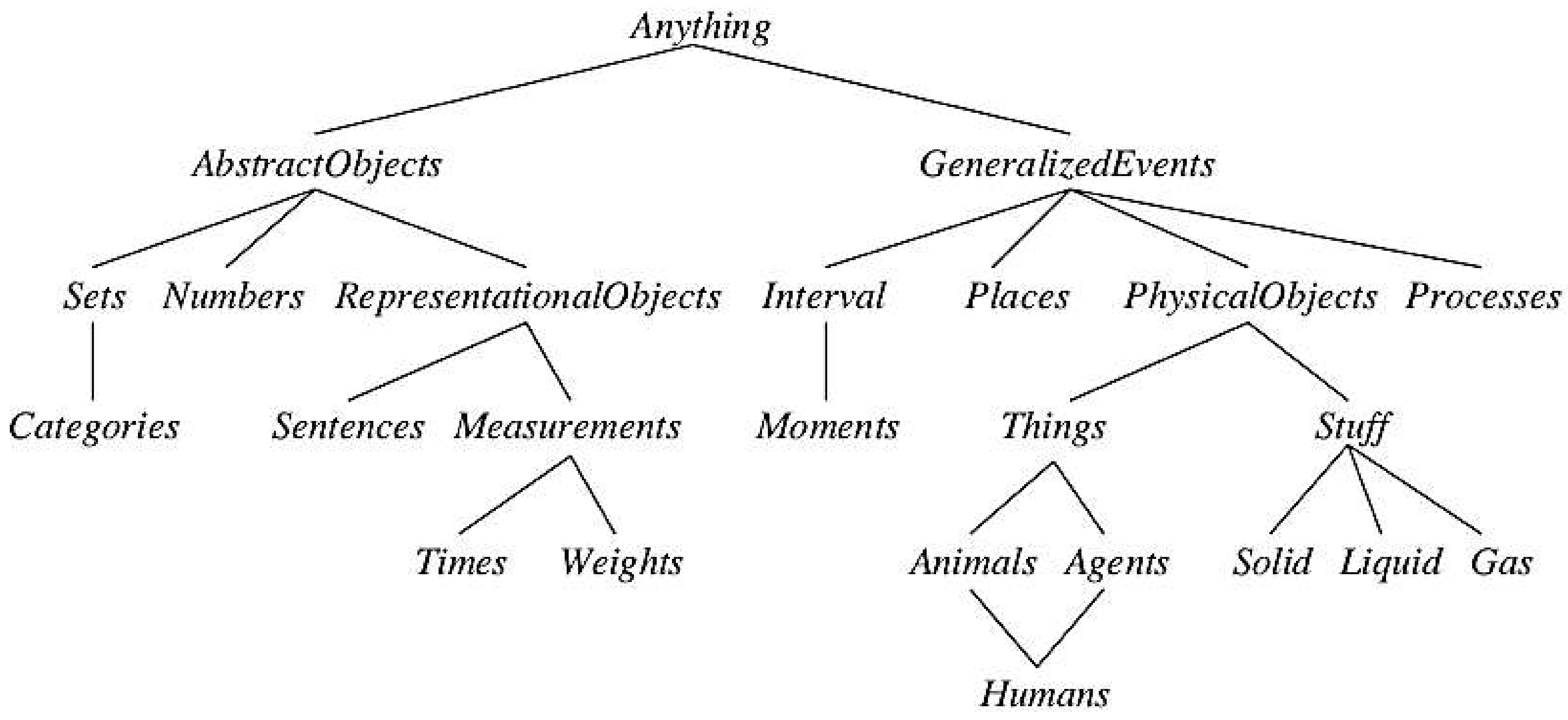
At step-5, we can infer the fact **Enemy(A, America)** from **Hostile(A)** which satisfies Rule-6. And hence all the statements are proved true using backward chaining.



Ontological Engineering

- ✓ Knowledge is the information about a domain that is used to solve problems of a domain.
- ✓ As part of designing a program to solve problems, we must define how the knowledge is represented.
- ✓ Complex domains require more general and flexible representations of concepts like events, time, physical object and beliefs of a domain.
- ✓ Representing these abstract concepts is called **Ontological Engineering**
- ✓ The general frame work of concepts is called **upper ontology,** because of the convention of drawing graphs with the general concepts at the top and the more specific concepts below them, as in Figure

Upper Ontology



General representation of ontology of world

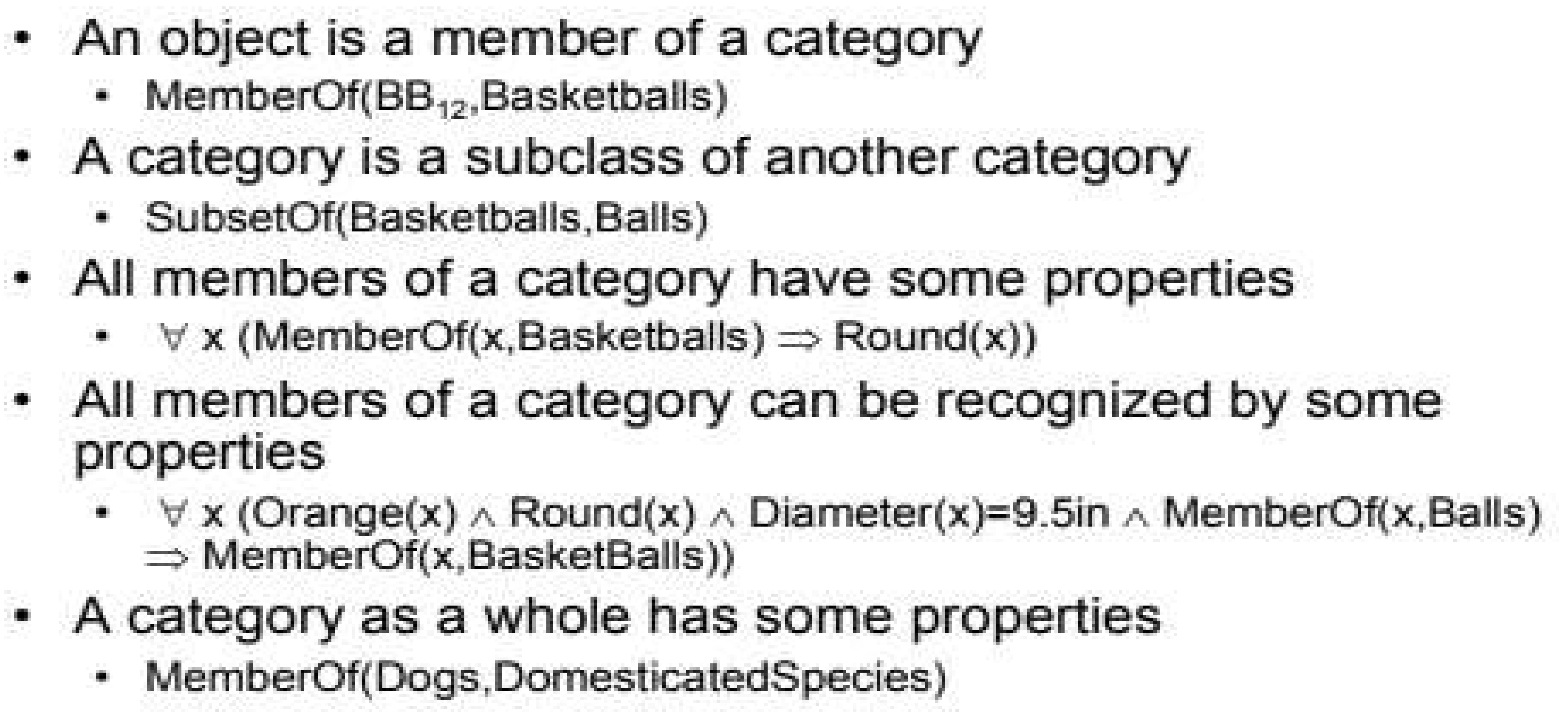
CATEGORIES AND OBJECTS

- The organization of objects into **categories** is a vital part of knowledge representation.
- Although interaction with the world takes place at the level of individual objects, *much reasoning takes place at the level of categories*.
- For example, a shopper would normally have the goal of buying a basketball, rather than a *particular* basketball such as BB9.
- Categories also serve to make predictions about objects once they are classified.
- There are two ways of representing categories in first-order logic:
  - Predicates : eg.Basketball (b),
  - Objects: eg.Basketballs

- Categories serve to organize and simplify the knowledge base through **inheritance**.
- If we say that all instances of the category Food are edible, and if we assert that Fruit is a subclass of Food and Apples is a subclass of Fruit, then we can infer that every apple is edible.
- We say that the individual apples **inherit** the property of edibility, in this case from their membership in the Food category.

First-order logic and categories

- First-order logic makes it easy to state facts about categories, either by relating objects to categories or by quantifying over their members.



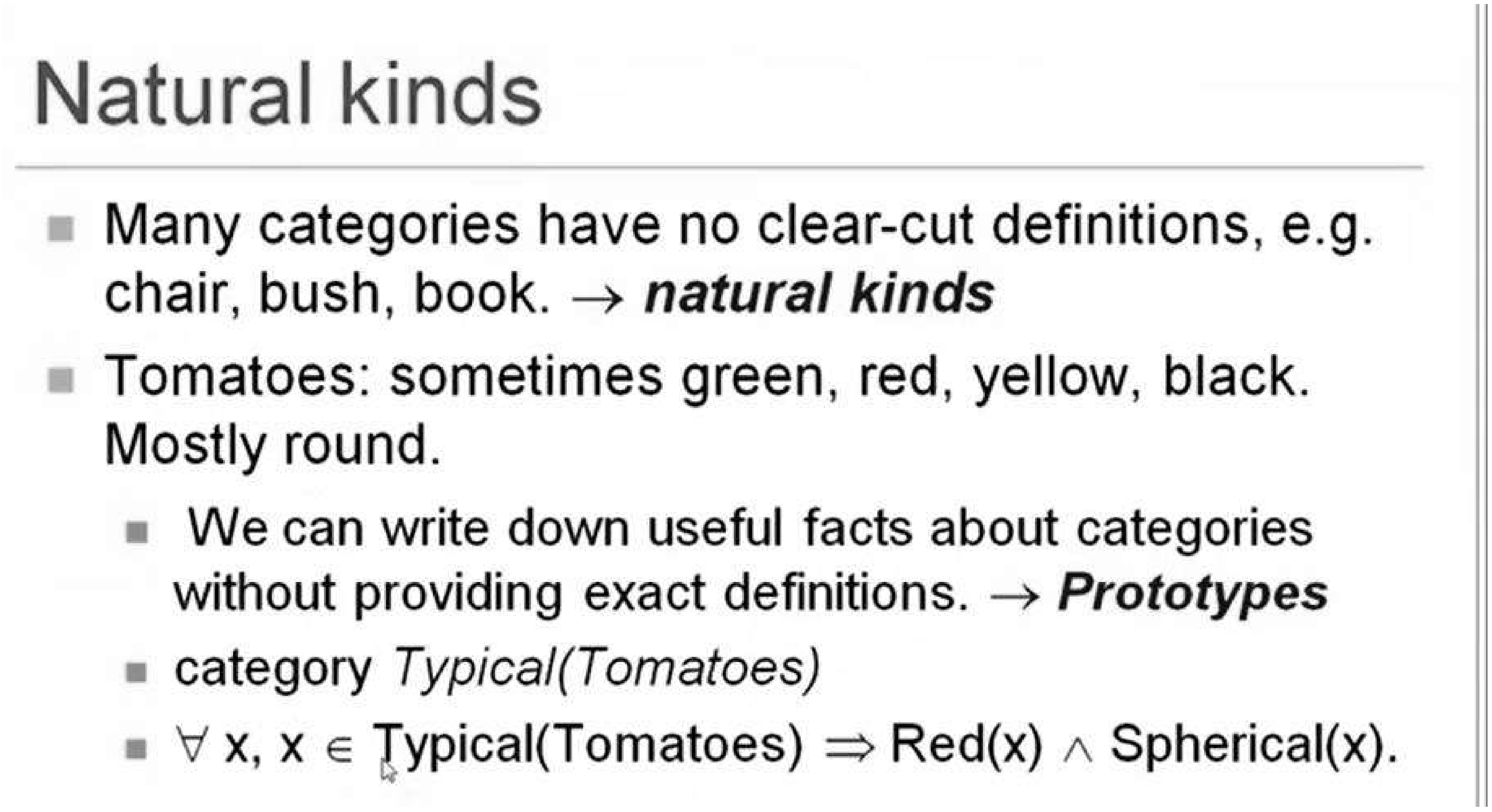
- An object is a member of a category
  - MemberOf(BB_{12},Basketballs)
- A category is a subclass of another category
  - SubsetOf(Basketballs,Balls)
- All members of a category have some properties
  - $\forall$ x (MemberOf(x,Basketballs) $\Rightarrow$ Round(x))
- All members of a category can be recognized by some properties
  - $\forall$ x (Orange(x) $\wedge$ Round(x) $\wedge$ Diameter(x)=9.5in $\wedge$ MemberOf(x,Balls) $\Rightarrow$ MemberOf(x,BasketBalls))
- A category as a whole has some properties
  - MemberOf(Dogs,DomesticatedSpecies)

Relations between categories

- Two or or more categories are ***disjoint*** if they have no members in common:
  - Disjoint(s)$\Leftrightarrow$($\forall$ c_1,c_2 $c_1 \in s \wedge c_2 \in s \wedge c_1 \neq c_2 \Rightarrow$ Intersection(c_1,c_2) $=\varnothing$)
  - Example; Disjoint({animals, vegetables})
- A set of categories *s* constitutes an ***exhaustive decomposition*** of a category *c* if all members of the set *c* are covered by categories in *s*:
  - E.D.(s,c) $\Leftrightarrow$ ($\forall$ i i $\in$ c $\Rightarrow \exists$ c_2 $c_2 \in s \wedge i \in c_2$)
  - Example: ExhaustiveDecomposition ({Americans, Canadian, Mexicans},NorthAmericans)

Relations between categories

- A ***partition*** is a disjoint exhaustive decomposition:
  - Partition(s,c) $\Leftrightarrow$ Disjoint(s) $\wedge$ E.D.(s,c)
  - Example: Partition({Males,Females},Persons).
- Is ({Americans,Canadian, Mexicans}, NorthAmericans) a partition? • No! There might be dual citizenships.
- Categories can be defined by providing necessary and sufficient conditions for membership
  - $\forall$ x Bachelor(x) $\Leftrightarrow$ Male(x) $\wedge$ Adult(x) $\wedge$ Unmarried(x)



Natural kinds

- Many categories have no clear-cut definitions, e.g. chair, bush, book. → ***natural kinds***
- Tomatoes: sometimes green, red, yellow, black. Mostly round.
  - We can write down useful facts about categories without providing exact definitions. → ***Prototypes***
  - category *Typical(Tomatoes)*
  - $\forall$ x, x $\in$ Typical(Tomatoes) $\Rightarrow$ Red(x) $\wedge$ Spherical(x).

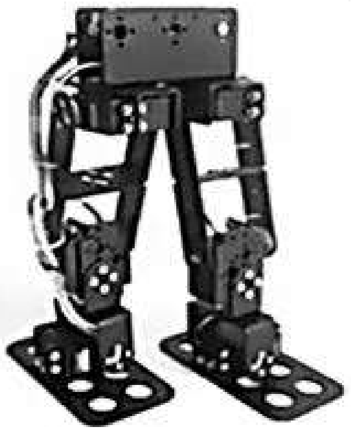
Physical composition

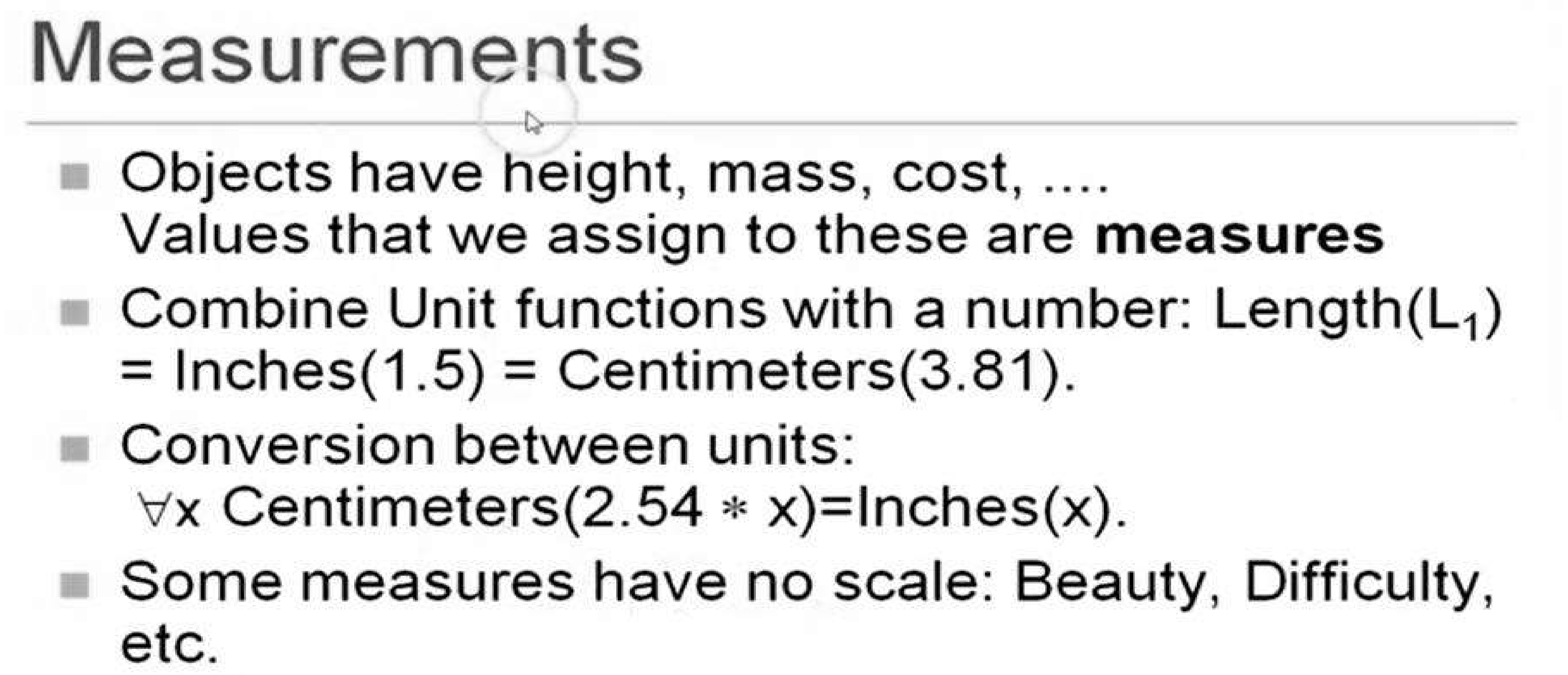
- One object may be part of another:
  - PartOf(Bucharest,Romania)
  - PartOf(Romania,EasternEurope)
  - PartOf(EasternEurope,Europe)
- The PartOf predicate is transitive (and irreflexive), so we can infer that PartOf(Bucharest,Europe)
- More generally:
  - $\forall$ x PartOf(x,x)
  - $\forall$ x,y,z PartOf(x,y) $\wedge$ PartOf(y,z) $\Rightarrow$ PartOf(x,z)

Physical composition

- Often characterized by structural relations among parts.
  - E.g. Biped(a) $\Rightarrow$

$$(\exists l_1, l_2, b)(Leg(l_1) \wedge Leg(l_2) \wedge Body(b) \wedge$$
$$PartOf(l_1, a) \wedge PartOf(l_2, a) \wedge PartOf(b, a) \wedge$$
$$Attached(l_1, b) \wedge Attached(l_2, b) \wedge$$
$$l_1 \neq l_2 \wedge (\forall l_3)(Leg(l_3) \Rightarrow (l_3 = l_1 \vee l_3 = l_2)))$$





Measurements

- Objects have height, mass, cost, ….
  Values that we assign to these are **measures**
- Combine Unit functions with a number: Length(L_1) = Inches(1.5) = Centimeters(3.81).
- Conversion between units:
  $\forall x$ Centimeters(2.54 $*$ x)=Inches(x).
- Some measures have no scale: Beauty, Difficulty, etc.

- Measures can be used to describe objects as follows:
  Diameter (Basketball)=Inches(9.5) .
  ListPrice(Basketball)=\$(19) .
  $d \in$ Days $\Rightarrow$ Duration(d)=Hours(24) .

Measurements

$$e_1 \in Exercises \land e_2 \in Exercises \land Wrote(Norvig, e_1) \land Wrote(Russell, e_2) \Rightarrow$$
$$\quad Difficulty(e_1) > Difficulty(e_2)\ .$$
$$e_1 \in Exercises \land e_2 \in Exercises \land Difficulty(e_1) > Difficulty(e_2) \Rightarrow$$
$$\quad ExpectedScore(e_1) < ExpectedScore(e_2)\ .$$

Objects

- the real world can be seen as consisting of primitive objects(e.g., atomic particles)and composite objects built from them.
- Stuff   ex: butter
- Things ex: dog
- some properties are intrinsic: they belong to the very substance of the object, rather than to the objects as a whole
- extrinsic properties-weight, length, shape and so on -are not retained under subdivision.
- A category of objects that includes in its definition only intrinsic properties is then a substance, or mass noun; a class that includes any extrinsic properties in its definition is a count noun

- The category **Stuff** is the most general substance category, specifying no intrinsic properties.
- The category **Thing** is the most general discrete object category, specifying no extrinsic properties.
- Ex:
  - any part of a butter-object is also a butter-object:

    $b \in Butter \wedge PartOf(p, b) \Rightarrow p \in Butter$.
  - We can now say that butter melts at around 30 degrees centigrade:

    $b \in Butter \Rightarrow MeltingPoint(b, Centigrade(30))$

Events

- situation calculus represents actions and their effects
- situation calculus- it was designed to describe a world in which actions are discrete, instantaneous and happen one at a time.
- Event calculus-based on points of time rather than on situations
- event calculus reifies **fluents** and **events**
- the fluent
  - At(shankar, berkeley): is an object that refers to the fact of shankar being in Berkeley, but does not by itself say anything about whether it is true.
- To assert that a fluent is actually true at some point in time we use the predicate T, as in
- T(At(shankar, Berkeley),t).

- Events are described as instances of event categories
- The event E1 of shankar flying from San Francisco to Washington D.C. is described as
- $E1 \in Flyings \wedge Flyer(E1, Shankar) \wedge Origin(E1, SF) \wedge Destination(E1,DC)$.
- we can define an alternative three-argument version of the category of flying events and say
- $E1 \in Flyings(Shankar, SF, DC)$.

- The complete set of predicates for one version of the event calculus is
- T(f, t) Fluent f is true at time t
- Happens(e, i) Event e happens over the time interval i
- Initiates(e, f, t) Event e causes fluent f to start to hold at time t
- Terminates(e, f, t) Event e causes fluent f to cease to hold at time t
- Clipped(f, i) Fluent f ceases to be true at some point during time interval i
- Restored (f, i) Fluent f becomes true sometime during time interval I
- For example, we can say that the only way a wumpus-world agent gets an arrow is at the start, and the only way to use up an arrow is to shoot it:
  - Initiates(e, HaveArrow(a), t) ⇔ e = Start
  - Terminates(e, HaveArrow(a), t) ⇔ e ∈ Shootings(a)

Process

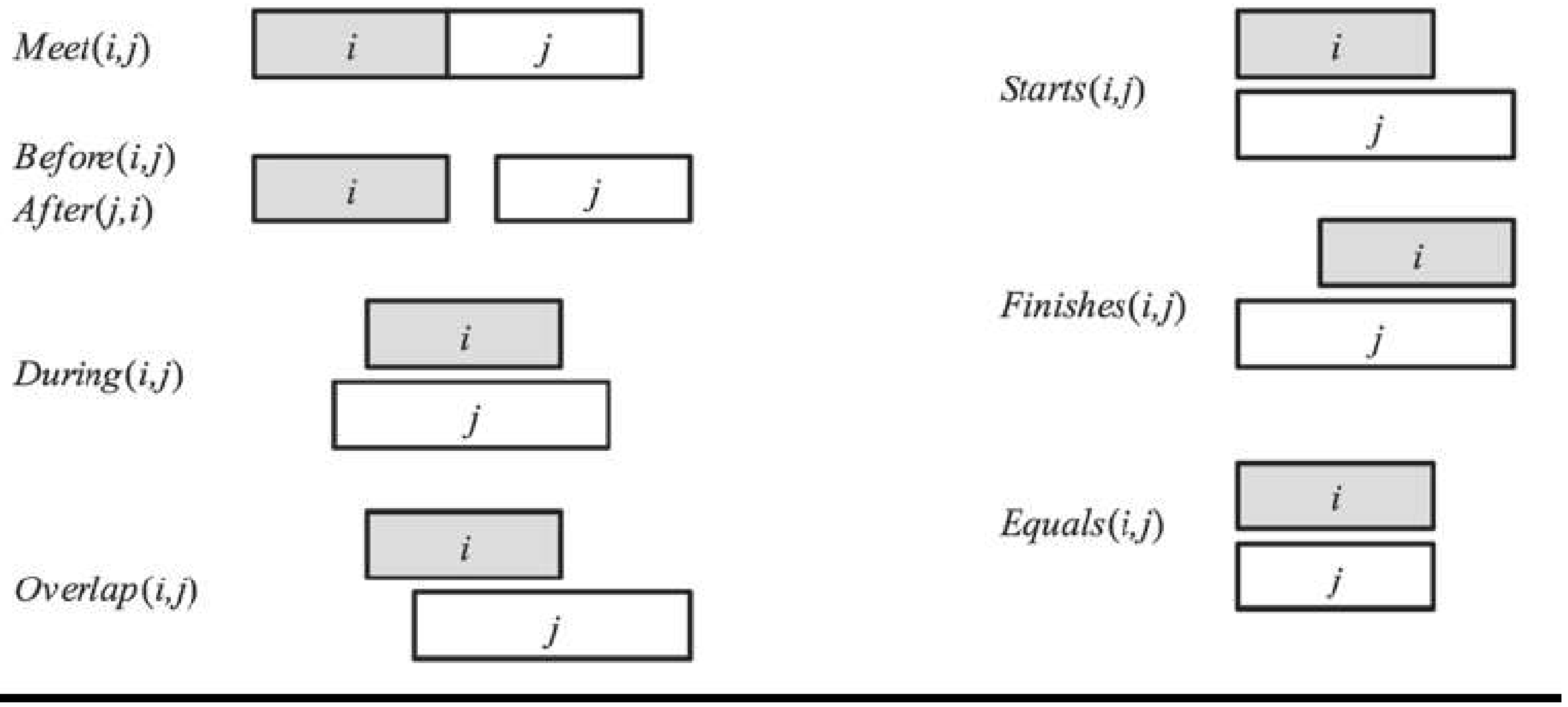
- If any event e that happens over an interval also happens at subinterval, then such categories of events are called process or liquid event categories.
- Any process e that happens over an interval also happens over any subinterval
- $(e \in Processes) \land Happens(e, (t1, t4)) \land (t1 < t2 < t3 < t4) \Rightarrow$
  $Happens(e, (t2, t3))$

Time intervals

- We will consider two kinds of time intervals: **moments** and **extended intervals**.
- The distinction is that only moments have zero duration:
- Partition({Moments, ExtendedIntervals}, Intervals)
- i ∈ Moments ⇔ Duration(i)=Seconds(0) .
- The functions **Begin** and **End** pick out the earliest and latest moments in an interval, and the function **Time** delivers the point on the time scale for a moment.(measure it in seconds, the moment at midnight (GMT) on January 1, 1900, has time 0)
- The function **Duration** gives the difference between the end time and the start time.
- Interval (i) ⇒ Duration(i)=(Time(End(i)) − Time(Begin(i))) .
- Time(Begin(AD1900))=Seconds(0) ;Time(Begin(AD2001))=Seconds(3187324800)
- Time(End(AD2001))=Seconds(3218860800) ;Duration(AD2001)=Seconds(31536000)

- a function Date, which takes six arguments (hours, minutes, seconds, day, month, and year) and returns a time point:
- Time(Begin(AD2001))=Date(0, 0, 0, 1, Jan, 2001)
- Date(0, 20, 21, 24, 1, 1995)=Seconds(3000000000) .
- The complete set of interval relations, as proposed by Allen (1983), is shown graphically in Figure and logically below:
- Meet(i, j) ⇔ End(i)=Begin(j)
- Before(i, j) ⇔ End(i) < Begin(j)
- After (j, i) ⇔ Before(i, j)
- During(i, j) ⇔ Begin(j) < Begin(i) < End(i) < End(j)
- Overlap(i, j) ⇔ Begin(i) < Begin(j) < End(i) < End(j)

Begins(i, j) ⇔ Begin(i) = Begin(j)
Finishes(i, j) ⇔ End(i) = End(j)
Equals(i, j) ⇔ Begin(i) = Begin(j) ∧ End(i) = End(j)



Predicates on time intervals.

Mental Events and Mental Objects

- Knowledge about one’s own knowledge and reasoning processes is useful for controlling inference.
- The agent should know what are in its knowledge base and what are not
- Knowledge about the knowledge of other agents is also important;
- What we need is a model of the mental objects that are in someone’s head (or something’s knowledge base) and of the mental processes that manipulate those mental objects.
- the **propositional attitudes** that an agent can have toward mental objects: attitudes such as Believes, Knows, Wants, Intends, and Informs.
- For eg. suppose we try to assert that Lois knows that Superman can fly:
- Knows(Lois, CanFly(Superman))

- if it is true that Superman is Clark Kent, then
- (Superman = Clark) ∧ Knows(Lois , CanFly(Superman))|= Knows(Lois, CanFly(Clark)) .
- if our agent knows that 2 + 2 = 4 and 4 < 5, then we want our agent to know that 2 + 2 < 5. This property is called **referential transparency**
- For propositional attitudes like *believes* and *knows*, we would like to have referential opacity
- **Modal logic** is designed to address this problem.
- Modal logic includes special modal operators that take sentences (rather than terms) as arguments.
- For example, "*A* knows *P*" is represented with the notation $\mathbf{K}_A P$, where **K** is the modal operator for knowledge

- we will need a more complicated model, one that consists of a collection of **possible worlds** rather than just one true world.
- The worlds are connected in a graph by **accessibility relations**, one relation for each modal operator.
- We say that world w1 is accessible from world w0 with respect to the modal operator $\mathbf{K}_A$ if everything in w1 is consistent with what A knows in w0, and we write this as Acc($\mathbf{K}_A$,w0,w1).
- In general, a knowledge atom $\mathbf{K}_A P$ is true in world w if and only if P is true in every world accessible from w.
- For example, we can say that, even though Lois doesn't know whether Superman's secret identity is Clark Kent, she does know that Clark knows:

$$\mathbf{K}_{Lois}[\mathbf{K}_{Clark}\, Identity(Superman, Clark) \vee \mathbf{K}_{Clark} \neg Identity(Superman, Clark)]$$

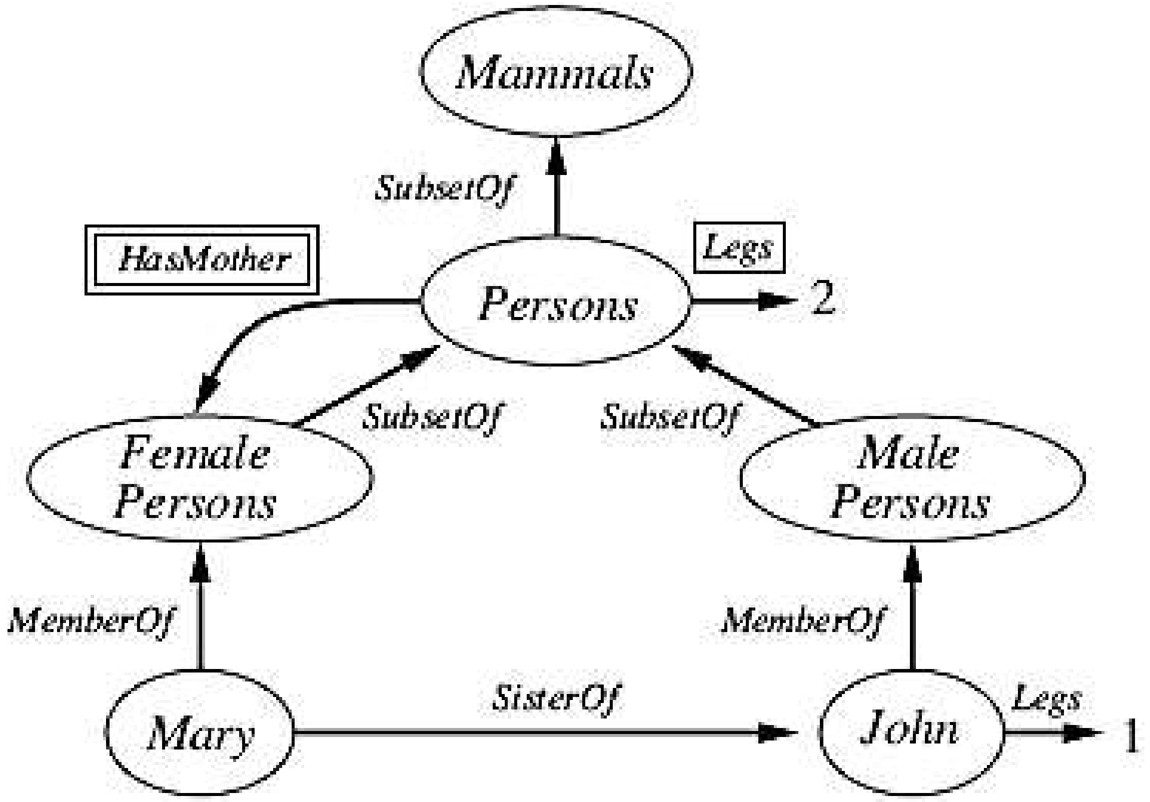
Reasoning Systems For Categories

- Categories are the primary building blocks of large-scale knowledge representation schemes.
- There are two systems specially designed for organizing and reasoning with categories.
- **semantic networks** provide graphical aids for visualizing a knowledge base and efficient algorithms for inferring properties of an object on the basis of its category membership;
- **description logics** provids a formal language for constructing and combining category definitions and efficient algorithms for deciding subset and superset relationships between categories.

Semantic Networks:

- a graphical notation of nodes and edges called **existential graphs.**
- There are many variants of semantic networks, but all are capable of representing individual objects, categories of objects, and relations among objects.
- A typical graphical notation displays object or category names in ovals or boxes, and connects them with labeled links
- For example, Figure has a MemberOf link between Mary and FemalePersons ,corresponding to the logical assertion

  Mary ∈ FemalePersons
- similarly, the SisterOf link between Mary and John corresponds to the assertion SisterOf (Mary, John).
- We can connect categories using SubsetOf links, and so on.

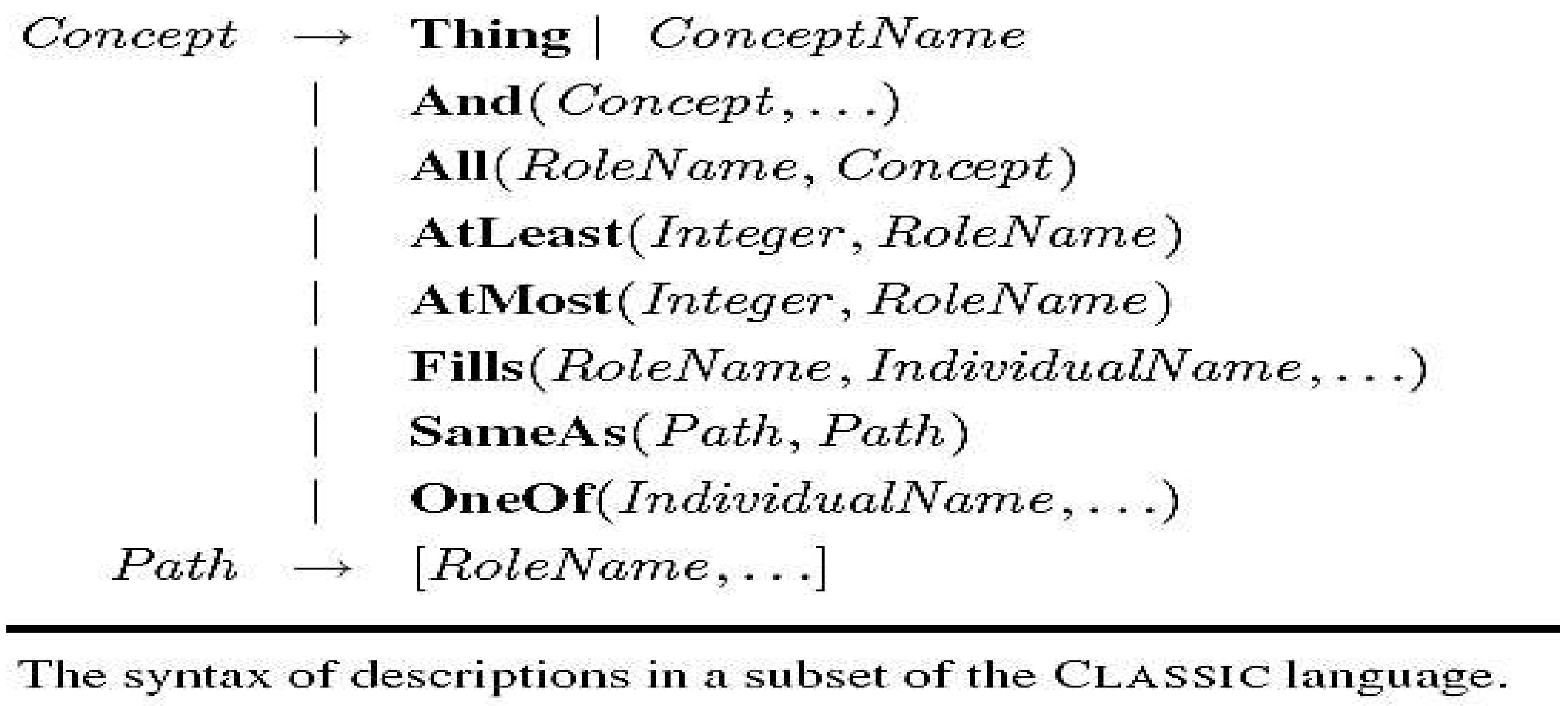


A semantic network with four objects (John, Mary, 1, and 2) and four categories. Relations are denoted by labeled links.

- we have used a special notation—the double-boxed link—in Figure
- $\forall x\ x \in Persons \Rightarrow [\forall\ y\ HasMother(x, y) \Rightarrow y \in FemalePersons\ ]$
- We might also want to assert that persons have two legs—that is,
- $\forall x, x \in Persons \Rightarrow Legs(x, 2)$.
- the single-boxed link in Figure is used to assert properties of every member of a category.
- The semantic network notation makes it convenient to perform **inheritance** reasoning

Description logics

- **Description logics** are notations that are designed to make it easier to describe definitions and properties of categories.
- The principal inference tasks for description logics are **subsumption** (checking if one category is a subset of another by comparing their definitions) and **classification** (checking whether an object belongs to a category)
- Some systems also include **consistency** of a category definition—whether the membership criteria are logically satisfiable
- The CLASSIC language is a typical description logic. The syntax of CLASSIC descriptions is shown in Figure



$$
\begin{array}{rcl}
\mathit{Concept} & \rightarrow & \textbf{Thing} \mid \mathit{ConceptName} \\
& \mid & \textbf{And}(\mathit{Concept}, \ldots) \\
& \mid & \textbf{All}(\mathit{RoleName}, \mathit{Concept}) \\
& \mid & \textbf{AtLeast}(\mathit{Integer}, \mathit{RoleName}) \\
& \mid & \textbf{AtMost}(\mathit{Integer}, \mathit{RoleName}) \\
& \mid & \textbf{Fills}(\mathit{RoleName}, \mathit{IndividualName}, \ldots) \\
& \mid & \textbf{SameAs}(\mathit{Path}, \mathit{Path}) \\
& \mid & \textbf{OneOf}(\mathit{IndividualName}, \ldots) \\
\mathit{Path} & \rightarrow & [\mathit{RoleName}, \ldots]
\end{array}
$$

The syntax of descriptions in a subset of the CLASSIC language.

For example, to say that bachelors are unmarried adult males we would write

Bachelor = And(Unmarried, Adult ,Male) .

The equivalent in first-order logic would be

Bachelor (x) ⇔ Unmarried(x) ∧ Adult(x) ∧ Male(x) .

- Any description in CLASSIC can be translated into an equivalent first-order sentence, but some descriptions are more straightforward in CLASSIC
- For example, to describe the set of men with at least three sons who are all unemployed and married to doctors, and at most two daughters who are all professors in physics or math departments, we would use

```
And(Man, AtLeast(3, Son), AtMost(2, Daughter ),
All(Son, And(Unemployed,Married, All(Spouse, Doctor ))),
All(Daughter , And(Professor , Fills(Department , Physics,Math))))
```

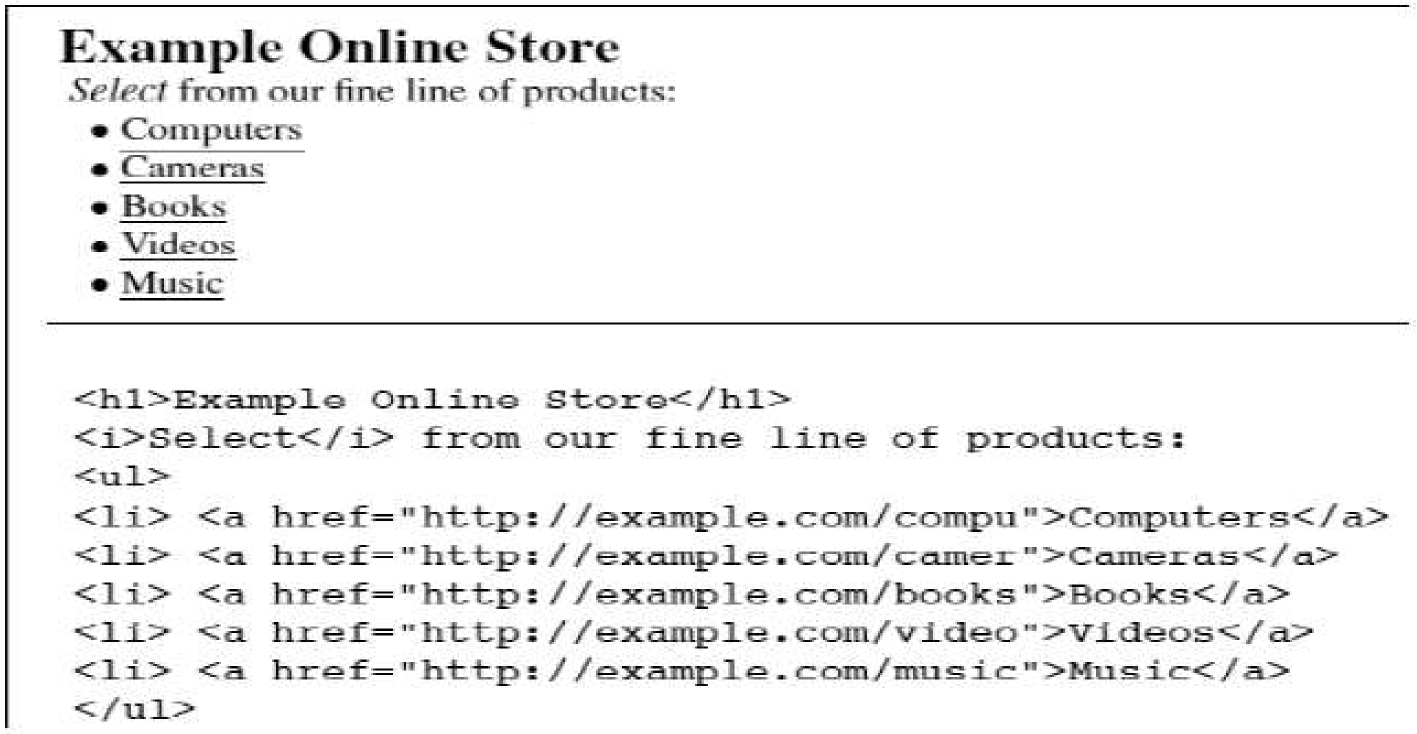
REASONING WITH DEFAULT INFORMATION

- **Circumscription** can be seen as a more powerful and precise version of the closed world assumption.
- The idea is to specify particular predicates that are assumed to be “as false as possible”—that is, false for every object except those for which they are known to be true.
- For example, suppose we want to assert the default rule that birds fly.
- We would introduce a predicate, say Abnormal1(x), and write
- $Bird(x) \land \neg Abnormal1(x) \Rightarrow Flies(x)$.
- If we say that Abnormal 1 is to be **circumscribed**, a circumscriptive reasoner is entitled to assume ¬Abnormal 1(x) unless Abnormal 1(x) is known to be true.
- This allows the conclusion Flies(Tweety) to be drawn from the premise Bird(Tweety), but the conclusion no longer holds if Abnormal1(Tweety) is asserted.

- **Default logic** is a formalism in which **default rules** can be written to generate contingent, nonmonotonic conclusions. A default rule looks like this:
- Bird(x) : Flies(x)/Flies(x) .
- This rule means that if Bird(x) is true, and if Flies(x) is consistent with the knowledge base, then Flies(x) may be concluded by default.
- In general, a default rule has the form
- P : J1, . . . , Jn/C
- where P is called the prerequisite, C is the conclusion, and Ji are the justifications.
- if any one of them can be proven false, then the conclusion cannot be drawn.
- Any variable that appears in Ji or C must also appear in P

The Internet Shopping world:

- In this section , we will create a shopping research agent that helps a buyer find product offers on the Internet .
- The shopping agent is given a product description by the buyer and has the task of producing a list of Web pages that offer such a product for sale and ranking for the best.



Example Online Store

Select from our fine line of products:

- Computers
- Cameras
- Books
- Videos
- Music

```
<h1>Example Online Store</h1>
<i>Select</i> from our fine line of products:
<ul>
<li> <a href="http://example.com/compu">Computers</a>
<li> <a href="http://example.com/camer">Cameras</a>
<li> <a href="http://example.com/books">Books</a>
<li> <a href="http://example.com/video">Videos</a>
<li> <a href="http://example.com/music">Music</a>
</ul>
```

The above figure shows a web page and a corresponding HTML character string.

- The Agents first task is to find relevant product offers.
- Let us consider query be a product description that the user types in (e.g.”laptop”);
- then a page is relevant offer for query,if the page is relevant and the page is indeed an offer. Also keep track of URL associated with the page.
- RelevantOffer(page,url,query) ⇔Relevant(page,url,query)^Offer(page).
- We can say a page is an offer if it contains the word ”buy” or “price” within an HTML link or form on the page.
- If the page contains a string of the form “<a...buy...</a>” then it is an offer, it could also be say “price” instead of “buy” or use “form” instead of “a”. We can write axioms for this:
- Offer(page) ⇔ (InTag(“a”,str,page)v InTag(“form”,str,page)) ^

  (In(“buy”,str) v In(“price”,str)).

- We need to find the relevant pages.
- The strategy is to start at the home page of an online store and consider all the pages that can be reached by the following relevant links.
- The Agent will have a knowledge of a number of stores, for example:

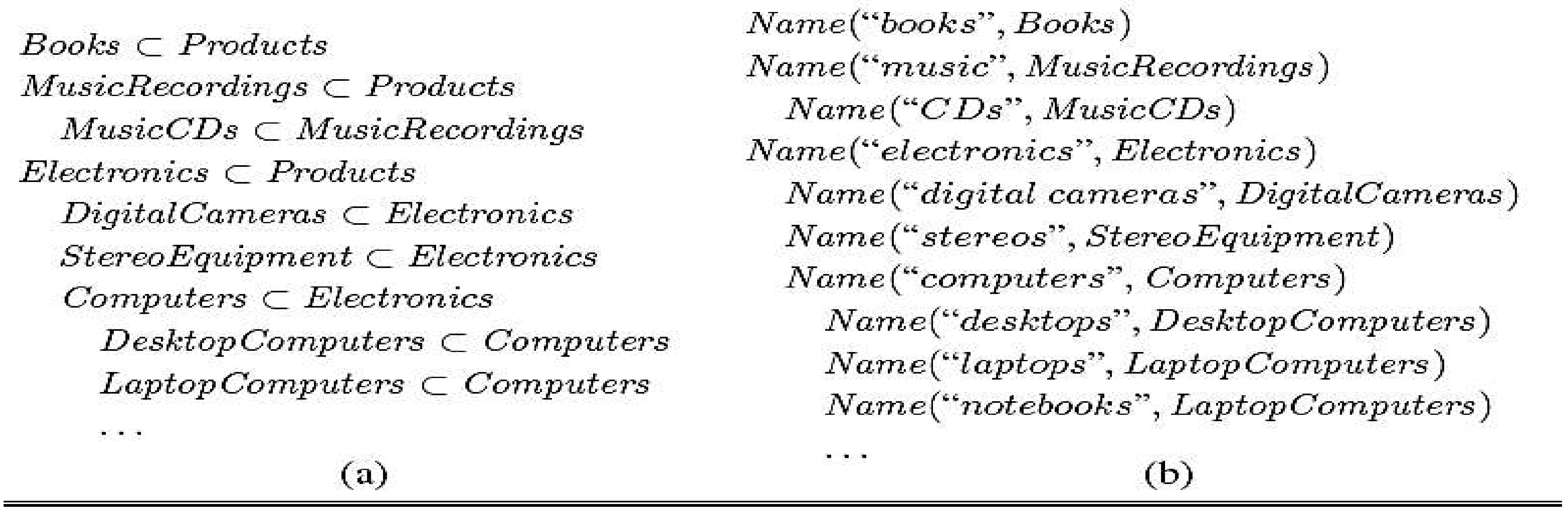
  Amazon ∈OnlineStores ∧ Homepage(Amazon, “amazon.com”) .

  Ebay ∈OnlineStores ∧ Homepage(Ebay, “ebay.com”) .

  ExampleStore ∈OnlineStores ∧ Homepage(ExampleStore, “example.com”)

- These stores classify their goods into product categories, and provide links to the Major categories from their home page.
- Minor categories can be reached by following a chain of relevant links, and eventually we will reach offers .
- A page is relevant to the query if it can be reached by a chain of relevant category links from a stores home page, and following one more link to the product offer :

- Relevant(page, query) ⇔ ∃ store, home store ∈OnlineStores ∧ Homepage(store, home) ∧ ∃url , url 2 RelevantChain(home, url 2, query) ∧ Link(url 2, url) ∧ page = Contents(url)
- A chain of links between two URLs, start and end ,is relevant to a description d if the anchor text of each link is relevant category name for d.
- The existence of the chain itself is determined by a recursive definition , with the empty chain (start = end)as the base case:
- First we need to relate strings to the categories they name.
- This is done by using the predicate Name(s,c), which says that string s is a name for category c –for example , we might assert that Name("laptops",LaptopComputers).



$Books \subset Products$
$MusicRecordings \subset Products$
  $MusicCDs \subset MusicRecordings$
$Electronics \subset Products$
  $DigitalCameras \subset Electronics$
  $StereoEquipment \subset Electronics$
  $Computers \subset Electronics$
    $DesktopComputers \subset Computers$
    $LaptopComputers \subset Computers$
    $\ldots$

(a)

$Name(\text{“books”}, Books)$
$Name(\text{“music”}, MusicRecordings)$
  $Name(\text{“CDs”}, MusicCDs)$
$Name(\text{“electronics”}, Electronics)$
  $Name(\text{“digital cameras”}, DigitalCameras)$
  $Name(\text{“stereos”}, StereoEquipment)$
  $Name(\text{“computers”}, Computers)$
    $Name(\text{“desktops”}, DesktopComputers)$
    $Name(\text{“laptops”}, LaptopComputers)$
    $Name(\text{“notebooks”}, LaptopComputers)$
    $\ldots$

(b)

Figure 12.9 (a) Taxonomy of product categories. (b) Names for those categories.

- Suppose the query is “laptops” , then RelevantCategoryName(query,text) is true when one of the following holds:
- The text and query name the same category—e.g.,”laptop computers” and “laptops”.
- The text names a super category such as “computers”.
- The text names a subcategory such as “ultralight notebooks”.
- The logical definition of RelevantCategoryName is as follows:
- RelevantCategoryName(query, text) ⇔ ∃ c1, c2 Name(query, c1) ∧ Name(text, c2) ∧ (c1 ⊆ c2 ∨ c2 ⊆ c1)

Comparing offers:

- To compare offers ,the agent must extract the relevant information –price ,speed ,disk size ,weight, and so on—from the offer pages.
- This can be difficult task with real web pages. A common way of dealing with this problem is to use programs called wrappers to extract information from a page.
- Consider given a page on the gen-store.com site with the text
- YVM ThinkBook 970. Our price: $1449.00
- Followed by various technical specifications , we would like a wrapper to extract information such as the following:
- ∃ c, offer c∈ LaptopComputers ∧ offer ∈ ProductOffers ∧ Manufacturer(c, IBM) ∧ Model (c, ThinkBook970) ∧ ScreenSize(c, Inches(14)) ∧ ScreenType (c, ColorLCD) ∧ MemorySize(c,Gigabytes(2)) ∧ CPUSpeed (c,GHz (1.2)) ∧ OfferedProduct(offer, c) ∧ Store(offer , GenStore) ∧ URL(offer , “example.com/computers/34356.html”) ∧ Price(offer , $(399)) ∧ Date(offer ,Today)

- The final task is to compare the offers that have been extracted . For example , consider these three offers:

  A : 1.4 GHz CPU, 2GB RAM, 250 GB disk, $299 .

  B : 1.2 GHz CPU, 4GB RAM, 350 GB disk, $500 .

  C : 1.2 GHz CPU, 2GB RAM, 250 GB disk, $399 .
- C is dominated by A; that is , A is cheaper and faster ,and they are otherwise the same.
- The shopping agent we have described here is a simple one ; many refinements are possible.

Unit-5

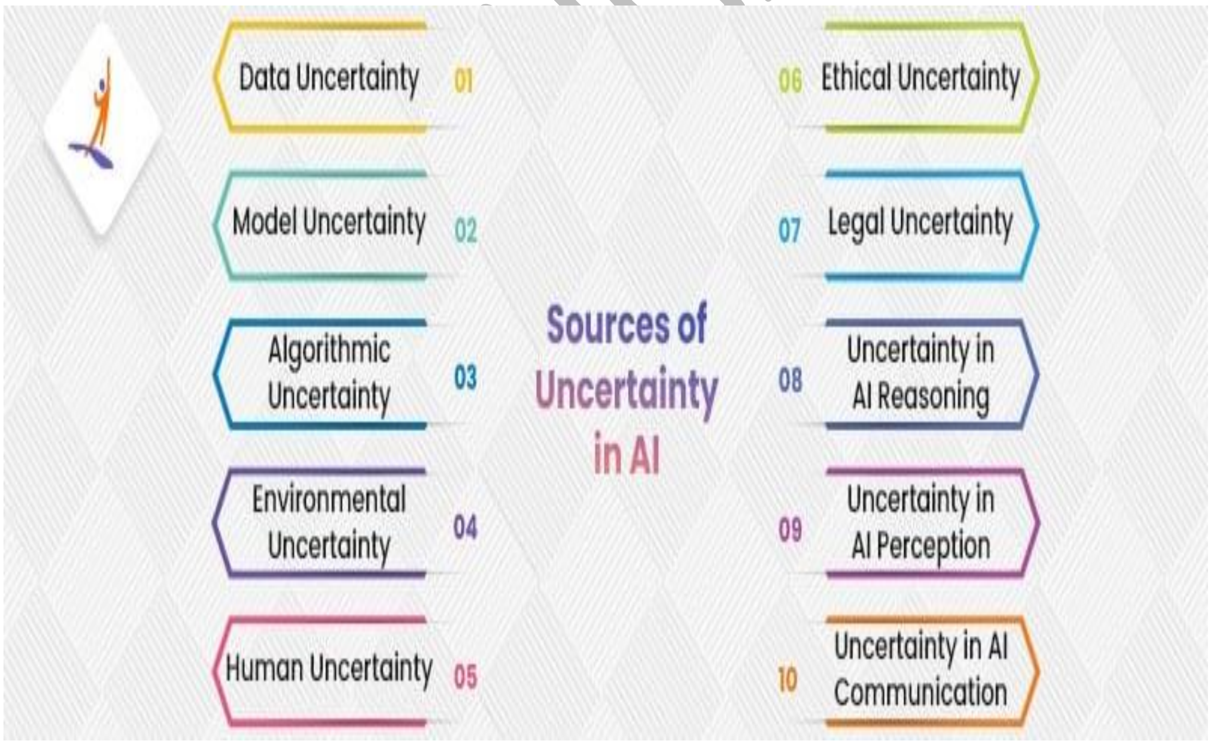
Quantifying Uncertainty

Acting Under Uncertainty

- Artificial intelligence (AI) uncertainty is when there's not enough information or ambiguity in data or decision-making. It is a fundamental concept in AI, as real-world data is often noisy and incomplete. AI systems must account for uncertainty to make informed decisions.
- AI deals with uncertainty by using models and methods that assign probabilities to different outcomes. Managing uncertainty is important for AI applications like self-driving cars and medical diagnosis, where safety and accuracy are key

Sources of Uncertainty in AI

There are several sources of uncertainty in AI that can impact the reliability and effectiveness of AI systems. Here are some common sources of uncertainty in AI:


Data Uncertainty 01
Model Uncertainty 02
Algorithmic Uncertainty 03
Environmental Uncertainty 04
Human Uncertainty 05
Sources of Uncertainty in AI
06 Ethical Uncertainty
07 Legal Uncertainty
08 Uncertainty in AI Reasoning
09 Uncertainty in AI Perception
10 Uncertainty in AI Communication

Data Uncertainty: AI models are trained on data, and the quality and accuracy of the data can affect the performance of the model. Noisy or incomplete data can lead to uncertain predictions or decisions made by the AI system.

Model Uncertainty: AI models are complex and can have various parameters and hyperparameters that need to be tuned. The choice of model architecture, optimization algorithm, and hyperparameters can significantly impact the performance of the model, leading to uncertainty in the results.

Algorithmic Uncertainty: AI algorithms can be based on different mathematical formulations, leading to different results for the same problem. For example, different machine learning algorithms can produce different predictions for the same dataset.

Environmental Uncertainty: AI systems operate in dynamic environments, and changes in the environment can affect the performance of the system. For example, an autonomous vehicle may encounter unexpected weather conditions or road construction that can impact its ability to navigate safely.

Human Uncertainty: AI systems often interact with humans, either as users or as part of the decision-making process. Human behaviour and preferences can be difficult to predict, leading to uncertainty in the use and adoption of AI systems.

Ethical Uncertainty: AI systems often raise ethical concerns, such as privacy, bias, and transparency. These concerns can lead to uncertainty in the development and deployment of AI systems, particularly in regulated industries.

Legal Uncertainty: AI systems must comply with laws and regulations, which can be ambiguous or unclear. Legal challenges and disputes can arise from the use of AI systems, leading to uncertainty in their adoption and implementation.

Uncertainty in AI Reasoning: AI systems use reasoning techniques to make decisions or predictions. However, these reasoning techniques can be uncertain due to the complexity of the problems they address or the limitations of the data used to train the models.

Uncertainty in AI Perception: AI systems perceive their environment through sensors and cameras, which can be subject to noise, occlusion, or other forms of interference. This can lead to uncertainty in the accuracy of the data used to train AI models or the effectiveness of AI systems in real-world applications.

Uncertainty in AI Communication: AI systems communicate with humans through natural language processing or computer vision. However, language and visual cues can be ambiguous or misunderstood, leading to uncertainty in the effective communication between humans and AI systems.

Types of Uncertainty in AI

Aleatoric Uncertainty: This type of uncertainty arises from the inherent randomness or variability in data. It is often referred to as "data uncertainty." For example, in a classification task, aleatoric uncertainty may arise from variations in sensor measurements or noisy labels.

Epistemic Uncertainty: Epistemic uncertainty is related to the lack of knowledge or information about a model. It represents uncertainty that can potentially be reduced with more data or better modelling techniques. It is also known as "model uncertainty" and arises from model limitations, such as simplifications or assumptions.

Parameter Uncertainty: This type of uncertainty is specific to probabilistic models, such as Bayesian neural networks. It reflects uncertainty about the values of model parameters and is characterized by probability distributions over those parameters.

Uncertainty in Decision-Making: Uncertainty in AI systems can affect the decision-making process. For instance, in reinforcement learning, agents often need to make decisions in environments with uncertain outcomes, leading to decision-making uncertainty.

Uncertainty in Natural Language Understanding: In natural language processing (NLP), understanding and generating human language can be inherently uncertain due to language ambiguity, polysemy (multiple meanings), and context-dependent interpretations.

Probability Notation

- Probabilistic notation refers to the symbols and conventions used to represent and manipulate probabilities and statistical concepts.
- This notation is fundamental in fields such as statistics, machine learning, and artificial intelligence

Basic Probabilistic Notations

Here are some key elements of probabilistic notation, which form the foundation for more advanced probabilistic models in AI:

Probability Notation:

Probability Notation	**Description**
$P(A)$	The probability of event A occurring
$P(A')$	The probability of event A not occurring
$P(A \cap B)$	The probability of both A and B occurring at the same time
$P(A \cup B)$	The probability of either A or B occurring
$P(A \cap B')$	The probability of A occurring but not B
$P(A' \cup B)$	The probability of either A not occurring or B occurring

Conditional Probability:

- **P(A | B)**: The probability of event A occurring given that event B has occurred. This is fundamental in AI for updating beliefs based on new evidence.
- **Bayes' Theorem**: $P(A|B)=P(B)P(B|A)\cdot P(A)P(A|B)=P(B)P(B|A)\cdot P(A)$, which provides a way to update probabilities based on new data.

Joint Probability:

- The probability of both A and B occurring, which can also be written as $P(A \cap B)$. This is essential for understanding the relationships between multiple variables.

Marginal Probability:

- The probability of event A **P(A)** occurring, regardless of other events. This is derived by summing or integrating over the joint probabilities of A with all other possible events.

Advanced Probabilistic Notations

Random Variables:

- **X**: A random variable representing a possible outcome.

- **P(X = x)**: The probability that the random variable X takes the value x.
- **P(X ≤ x)**: The probability that the random variable X takes a value less than or equal to x.

Probability Distributions:

- **Probability Mass Function (PMF)**: For discrete random variables, $P(X=x)P(X=x)$denotes the PMF.
- **Probability Density Function (PDF)**: For continuous random variables, $fX(x)fX(x)$ denotes the PDF.
- **Cumulative Distribution Function (CDF)**: $FX(x)=P(X\leq x)FX(x)=P(X\leq x)$ gives the cumulative probability up to x.

Expectation and Variance:

- **E[X]**: The expected value or mean of the random variable X.
- **Var(X)**: The variance of the random variable X, representing the spread of its possible values.

Covariance and Correlation:

- **Cov(X, Y):** The covariance between random variables X and Y, indicating the degree to which they change together.
- **Corr(X, Y)**: The correlation coefficient between X and Y, a normalized measure of their linear relationship.

Inference Using Full Joint Distributions

- Joint probability offers valuable insights into the likelihood of multiple events happening together. This helps us in several ways:

Co-occurrence: Joint probability helps us understand how likely it is for two or more events to happen at the same time. This is important for seeing how events are connected and the probability of them occurring together.

Risk Evaluation: In areas like finance and insurance, joint probability helps us assess the risk when multiple events overlap. For instance, it can estimate the chance of multiple financial instruments facing losses simultaneously.

Quality Check: Businesses can use joint probability to gauge the reliability and quality of their products or processes. It shows the likelihood of multiple defects or issues occurring at once, which allows for proactive quality improvement efforts.

Event Relationships: Joint probability can indicate if events are related or not. If joint probability significantly differs from the product of individual probabilities, it suggests events are connected, and the occurrence of one affects the likelihood of the other.

Decision Support: When businesses need to make choices involving multiple factors or events, joint probability provides a numerical foundation for decision-making. It helps assess how different variables together impact the desired outcome.

Resource Management: In situations with limited resources, understanding joint probability helps optimise resource allocation. For example, in supply chain management, it can estimate the chance of multiple supply chain disruptions happening at the same time, enabling better risk management strategies.

Formula for Joint Probability

For Independent Events

When events A and B are independent, meaning that the occurrence of one event does not impact the other, we use the multiplication rule:

$P(A \cap B) = P(A) \times P(B)$

Here, P(A) is the probability of occurrence of event A, P(B) is the probability of occurrence of event B, and $P(A \cap B)$ is the joint probability of events A and B.

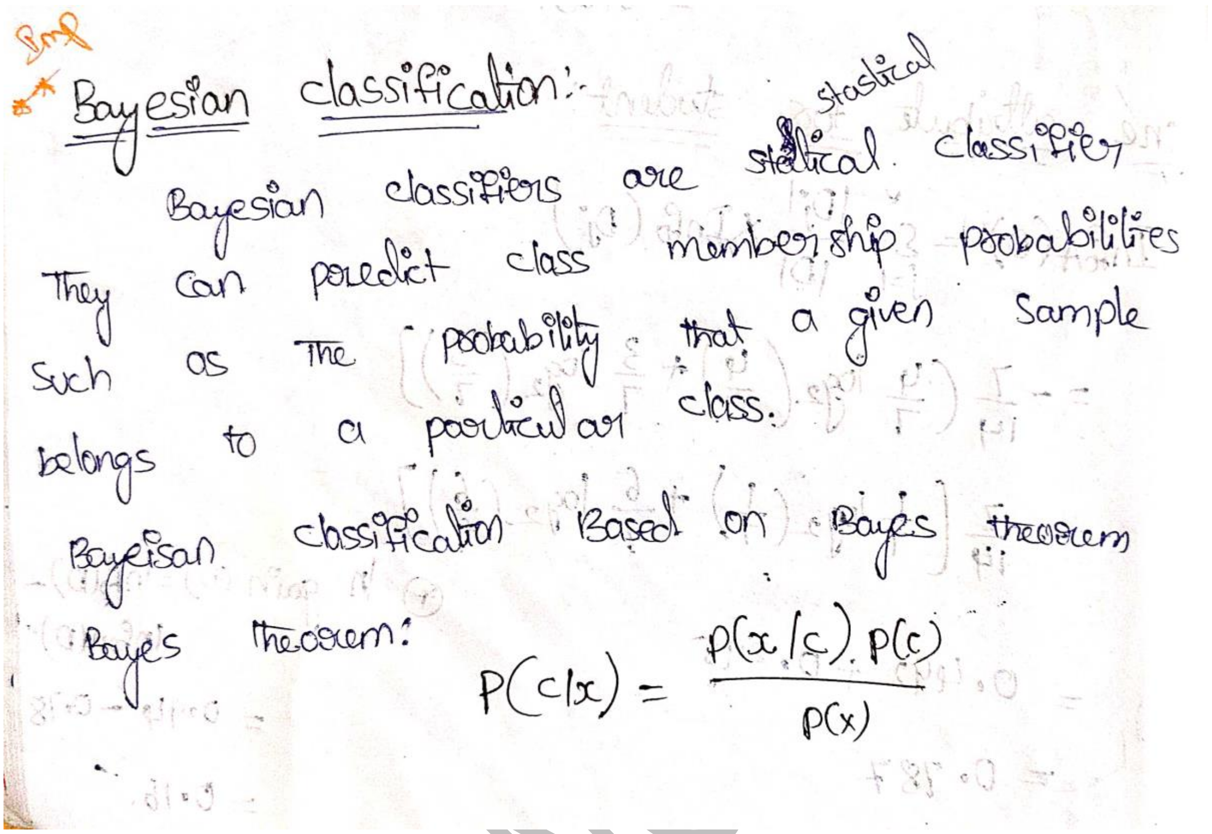
For Dependent Events

Events are often dependent on each other, meaning that one event's occurrence influences the likelihood of the other. Here, we employ a modified formula:

$P(A \cap B) = P(A) \times P(B|A)$

Here, P(A) is the probability of occurrence of event A, P(B|A) is the conditional probability of occurrence of event B when event A has already occurred, and $P(A \cap B)$ is the joint probability of events A and B.

Bayesian Classification:



Imp

* **Bayesian classification:**

Bayesian classifiers are stastical classifier. They can predict class membership probabilities such as the probability that a given sample belongs to a particular class.

Bayeisan classification Based on Bayes theorem

Bayes theorem:

$$P(c|x) = \frac{p(x/c).p(c)}{p(x)}$$

It finds postperior probability and poiority probabili-ty.
* It find the postprior probability of a class conditinal.

Algorithm:-

step 1:-
* let a 'D' be a training dataset of Data tuple associated with class lables.

step 2:-
* each tuple is represented by an N-dimensional vector $X = (x_1, x_2 ---x_n)$ where $x_1, x_2 ---x_n$ are values of attributes.
* Suppose that there are m no. of classes $C_1, C_2 ---C_m$ given a data tuple x the classifier is predict that $x \in$ the class having the highest posterior probability condition on X.
* This can be written mathematically where $j = 1, 2, ---m$ and $i \neq j$.

i.e., we maximize $P(C_i | x)$

Bayes theorem $P(C_i|x) = \frac{P(x|C_i)P(C_i)}{P(x)}$

step 3:-
As $P(x)$ is constant for all classes it is enough to maximize only that is numerical function has $P(x|C_i)$.

* If the class probability is not known.

$p(C_1) = p(C_2) = \cdots = p(C_m)$

so it is enough to maximize only $p(x/C_i)$

step 4:-

$$p(x/C_i) = \prod_{k=1}^{m} p(x_k | C_i)$$

$$= p(x_1/C_i) \times p(x_2|C_i) \times \cdots \times p(x_m|C_i)$$

(i) If A_k categorical $p(x_k|C_i)$ is number of tuples of class C_i in 'D' having the value small S_k for A_k / no. of tuples of C_i in D.

(ii) A_k is continuous $g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\,\sigma_{C_i}} \times e^{\frac{-(x-\mu)^2}{2\sigma^2}}$

where $g(x, \mu, \sigma)$ is the Gaussian (normal) density function for attribute A_k while μ and σ are the mean and standard deviation, respectively given the values for attributes A_k for training samples of class C.

step 5:-

The classifier predict the class lable is C_i if and only if $p(x/C_i)\, p(C_i) > p(x|C_j) \cdot p(C_j)$ for $1 \le j \le m$, $j \ne i$

Problem :-

RID	Age	Income	student	credit rating	buys Computer.
1	youth	high	NO	Avg	NO
2	"	"	"	Exce.	"
3	middle	"	"	Avg	yes
4	senior	medium	"	"	"
5	senior	low	yes	"	"
6	senior	"	yes	Exce	NO
7	middle	"	yes	"	yes
8	youth	medium	~~yes~~ NO	Avg	NO
9	youth	low	~~NO~~ yes	"	yes
10	senior	medium	yes	"	"
11	youth	"	"	Exce	"
12	middle	"	NO	Exce	"
13	middle	high	yes	Avg	yes
14	senior	medium	NO	Exce	NO

X = (age = "<= 30", income = "medium", student = "yes", credit - rating = "$\overset{\text{avg.}}{\text{fair}}$") predict the class labeled data tuple 'x' using bayesian classification algorithm.

Sol :- The prior probability of each class.

$P(\text{buys\_computer} = \text{"yes"}) = \frac{9}{14} = 0.6428$

$P(\text{buys - computer} = \text{"NO"}) = \frac{5}{14} = 0.3571$

young:-

$P(\text{age} <= 30 \mid \text{buys.computer} = \text{"yes"}) = \frac{2}{9} = 0.222$

$P(\text{age} <= 30 \mid \text{buys computer} = \text{"NO"}) = \frac{3}{5} = 0.600$

medium:-

$P(\text{income} = \text{"medium"} \mid \text{buys computer} = \text{"yes"}) = \frac{4}{9} = 0.444$

$P(\text{income} = \text{"medium"} \mid \text{buys computer} = \text{"NO"}) = \frac{2}{5} = 0.40$

student:-

$P(\text{student} = \text{"yes"} \mid \text{buys computer} = \text{"yes"}) = \frac{6}{9} = 0.666$

$P(\text{student} = \text{"yes"} \mid \text{buys computer} = \text{"NO"}) = \frac{1}{5} = 0.200$

credit - rating:-

$P(\text{credit\_rating} = \text{"avg"} \mid \text{buys computer} = \text{"yes"}) = \frac{6}{9} = 0.666$

$P(\text{"} = \text{"avg"} \mid \text{buys computer} = \text{"NO"}) = \frac{2}{5} = 0.400$

using the above probabilities:-

$P(x \mid \text{buys.computer} = \text{"yes"}) = 0.222 \times 0.444 \times 0.666 \times 0.666$
$= 0.0437$

$P(x \mid \text{buys.computer} = \text{"NO"}) = 0.600 \times 0.400 \times 0.200 \times 0.400$

$= 0.019$